User Guide

AWS Identity and Access Management



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Identity and Access Management: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is IAM? 1
Why should I use IAM? 3
Shared access to your AWS account 3
Granular permissions 3
Secure access to AWS resources for applications that run on Amazon EC2 3
Multi-factor authentication (MFA) 3
Identity federation 4
Identity information for assurance 4
PCI DSS Compliance 4
When do I use IAM? 4
When you are performing different job functions 5
When you are authorized to access AWS resources 5
When you sign-in as an IAM user 6
When you assume an IAM role 6
When you create policies and permissions 6
How do I manage IAM? 7
Use the AWS Management Console 8
AWS Command Line Tools 9
Use the AWS SDKs 11
Use the IAM Query API 11
How IAM works 12
Components of a request 13
How principals are authenticated 14
Authorization and permission policy basics 14
..... 15
Compare IAM identities and credentials 16
Terms 17
Difference between IAM users and users in IAM Identity Center 19
Federate users from an existing identity source 20
Different methods to provide user access 21
Support programmatic user access 25
How permissions and policies provide access management 26
Policies and accounts 26
Policies and users 27

Policies and IAM groups ... 27
Federated user sessions and roles ... 28
Identity-based and resource-based policies ... 28
Define permissions with ABAC authorization ... 29
Comparison of ABAC to the traditional RBAC model ... 30
Getting started ... 32
Setting up your AWS account ... 32
Viewing your AWS account ID ... 34
To view your AWS account ID ... 35
Using an alias for your AWS account ID ... 36
Creating an account alias ... 37
Deleting an account alias ... 38
Plan access to your AWS account ... 39
Identity sources ... 39
Access management ... 40
Use cases for IAM users ... 40
Use multi-factor authentication with your identities ... 53
Prepare for least-privilege permissions ... 54
Reviewing last accessed information for your AWS account ... 55
Generating a policy based on access activity ... 61
Using search to find IAM resources ... 64
Security best practices and use cases ... 68
Security best practices ... 68
Require human users to use federation with an identity provider to access AWS using temporary credentials ... 69
Require workloads to use temporary credentials with IAM roles to access AWS ... 70
Require multi-factor authentication (MFA) ... 71
Update access keys when needed for use cases that require long-term credentials ... 71
Follow best practices to protect your root user credentials ... 72
Apply least-privilege permissions ... 73
Get started with AWS managed policies and move toward least-privilege permissions ... 73
Use IAM Access Analyzer to generate least-privilege policies based on access activity ... 73
Regularly review and remove unused users, roles, permissions, policies, and credentials ... 73
Use conditions in IAM policies to further restrict access ... 74
Verify public and cross-account access to resources with IAM Access Analyzer ... 74

Use IAM Access Analyzer to validate your IAM policies for secure and functional permissions ... 74
Establish permissions guardrails across multiple accounts ... 75
Use permissions boundaries to delegate permissions management within an account ... 75
Root user best practices ... 75
Secure your root user credentials to prevent unauthorized use ... 77
Use a strong root user password to help protect access ... 77
Secure your root user sign-in with multi-factor authentication (MFA) ... 77
Don't create access keys for the root user ... 78
Use multi-person approval for root user sign-in wherever possible ... 78
Use a group email address for root user credentials ... 79
Restrict access to account recovery mechanisms ... 79
Secure your AWS Organizations account root user credentials ... 79
Monitor access and usage ... 80
Business use cases ... 82
Initial setup of example corp ... 82
Use case for IAM with Amazon EC2 ... 84
Use case for IAM with Amazon S3 ... 85
Tutorials ... 87
Delegate access across AWS accounts using roles ... 87
Considerations ... 89
Prerequisites ... 90
Create a role in the Destination Account ... 90
Grant access to the role ... 94
Test access by switching roles ... 96
Additional resources ... 101
Summary ... 102
Create a customer managed policy ... 102
Prerequisites ... 102
Step 1: Create the policy ... 103
Step 2: Attach the policy ... 104
Step 3: Test user access ... 104
Related resources ... 105
Summary ... 105
Use attribute-based access control (ABAC) ... 105
Tutorial overview ... 106

Prerequisites ... 108
Step 1: Create test users ... 108
Step 2: Create the ABAC policy ... 110
Step 3: Create roles ... 114
Step 4: Test creating secrets ... 115
Step 5: Test viewing secrets ... 118
Step 6: Test scalability ... 120
Step 7: Test updating and deleting secrets ... 122
Summary ... 124
Related resources ... 124
Use SAML session tags for ABAC ... 125
Permit users to manage their credentials and MFA settings ... 129
Prerequisites ... 130
Step 1: Create a policy to enforce MFA sign-in ... 131
Step 2: Attach policies to your test user group ... 132
Step 3: Test your user's access ... 132
Related resources ... 134
Create SAML IdP with CloudFormation ... 135
Prerequisites ... 135
Create the IdP ... 135
Verify the IdP ... 138
Delete resources ... 139
Template details ... 139
Template ... 140
Create SAML federated role with CloudFormation ... 144
Prerequisites ... 144
Create the federated role ... 144
Test the federated role ... 147
Delete resources ... 148
Template details ... 148
Template ... 148
Create SAML IdP and federated role with CloudFormation ... 155
Prerequisites ... 155
Create the resources ... 155
Test federation ... 159
Delete resources ... 159

Template details ... 160
Template ... 160
Identities ... 170
IAM root user ... 171
IAM users ... 171
IAM user groups ... 171
IAM roles ... 171
AWS account root user ... 171
Centrally manage root access for member accounts ... 172
Centralize root access ... 173
Perform a privileged task ... 177
MFA for the root user ... 183
Change the password ... 191
Reset a lost or forgotten root user password ... 193
Create access keys for the root user ... 195
Delete access keys for the root user ... 197
Tasks that require root user credentials ... 199
Additional resources ... 201
Users ... 202
How AWS identifies an IAM user ... 202
IAM users and credentials ... 203
IAM users and permissions ... 204
IAM users and accounts ... 205
IAM users as service accounts ... 205
How IAM users sign in to AWS ... 205
Create a user ... 210
View IAM users ... 211
Rename a user ... 214
Remove a user ... 216
Control user access to the console ... 220
Change user permissions ... 222
User passwords ... 230
Manage access keys ... 250
Multi-factor authentication ... 279
Service-specific credentials ... 356
Find unused credentials ... 366

Generate credential reports ... 370
IAM credentials for CodeCommit ... 377
Manage server certificates ... 380
User groups ... 386
Create IAM groups ... 388
View IAM groups ... 389
Edit users in IAM groups ... 391
Attach a policy to a user group ... 392
Rename a user group ... 393
Delete an IAM group ... 394
Roles ... 396
When to create an IAM user (instead of a role) ... 398
Roles terms and concepts ... 399
Additional resources ... 402
The confused deputy problem ... 403
Common scenarios ... 414
Role creation ... 427
Role management ... 494
Methods to assume a role ... 529
Identity providers and federation into AWS ... 690
Federation with IAM Identity Center ... 691
Federation with IAM ... 692
Federation with Amazon Cognito identity pools ... 692
Additional resources ... 693
Common scenarios ... 693
OIDC federation ... 698
SAML 2.0 federation ... 724
Federating AWS identities to external services ... 764
Common use cases ... 764
How It Works ... 765
Getting started with outbound identity federation ... 766
Understanding token claims ... 773
Controlling access with IAM policies ... 779
Federating AWS identities across AWS partitions ... 781
How cross-partition identity federation works ... 782
Setting up cross-partition identity federation ... 784

Controls for cross-partition identity federation ... 787
Auditing cross-partition identity federation ... 792
Temporary security credentials ... 796
AWS STS and AWS regions ... 797
Common scenarios for temporary credentials ... 797
Sample applications that use temporary credentials ... 799
Additional resources for temporary credentials ... 800
Compare AWS STS credentials ... 801
Service bearer tokens ... 804
Request temporary security credentials ... 805
Use temporary credentials with AWS resources ... 816
Permissions for temporary security credentials ... 821
Manage AWS STS in an AWS Region ... 858
AWS STS Regions and endpoints ... 863
Enable custom identity broker console access ... 868
Tags for IAM resources ... 883
Choose an AWS tag naming convention ... 884
Rules for tagging in IAM and AWS STS ... 885
Tag IAM users ... 888
Tag IAM roles ... 891
Tag customer managed policies ... 894
Tag OIDC identity providers ... 897
Tag IAM SAML identity providers ... 900
Tagging instance profiles ... 902
Tag server certificates ... 905
Tag virtual MFA devices ... 907
Tag account access manager instances ... 910
Pass session tags ... 912
Access management ... 928
Access management resources ... 929
Policies and permissions ... 930
Policy types ... 930
Policies and the root user ... 937
Overview of JSON policies ... 937
Grant least privilege ... 942
Managed policies and inline policies ... 943

Data perimeters ... 954
Permissions boundaries ... 960
Identity vs resource ... 974
Control access using policies ... 977
Control access to IAM users and roles using tags ... 990
Control access to AWS resources using tags ... 993
Cross account resource access ... 998
Forward access sessions ... 1004
Example policies ... 1007
Manage IAM policies ... 1089
Additional resources ... 1089
Create IAM policies ... 1090
Policy validation ... 1100
Policy testing ... 1101
Add or remove identity permissions ... 1114
Versioning IAM policies ... 1126
Edit IAM policies ... 1131
Delete IAM policies ... 1141
Refine permissions using access information ... 1147
Policy summaries ... 1734
Policy summary (list of services) ... 1735
Access levels in policy summaries ... 1746
Service summary (list of actions) ... 1749
Action summary (list of resources) ... 1754
Example policy summaries ... 1758
Permissions required to access IAM resources ... 1768
Permissions for administering IAM identities ... 1768
Permissions for working in the AWS Management Console ... 1770
Grant permissions across AWS accounts ... 1771
Permissions for one service to access another ... 1771
Required actions ... 1772
Example policies for IAM ... 1773
Temporary delegation ... 1777
Overview ... 1777
How temporary delegation works ... 1777
Initiate a temporary delegation request ... 1780

Review Temporary delegation requests ... 1783
Revoke Temporary delegation access ... 1784
Managing Permissions for Delegation Requests ... 1785
Notifications ... 1786
CloudTrail ... 1787
Temporary delegation for Partners ... 1789
Overview ... 1789
How it works ... 1777
Partner qualification ... 1790
Onboarding process ... 1790
Partner questionnaire ... 1791
Understanding your integration ... 1793
Understanding permissions ... 1796
Policy evaluation guidelines ... 1798
Policy templates ... 1801
Building your integration ... 1812
Account access manager ... 1821
What is account access manager? ... 1821
Why use account access manager? ... 1821
Key capabilities ... 1823
Getting started ... 1824
Prerequisites and considerations ... 1825
Enable account access manager ... 1825
Update firewalls and gateways ... 1826
Delete account access manager ... 1828
Manage access to AWS accounts ... 1829
Delegated administration ... 1830
Prepare your IAM roles ... 1833
Assign and remove access ... 1835
View or search assignments ... 1839
Sessions and access control ... 1841
Workforce access ... 1842
User portals ... 1842
Getting started with the account access portal ... 1843
User access to AWS accounts ... 1844
Security ... 1848

Identity and access management for account access manager ... 1849
Data protection in account access manager ... 1857
Logging and monitoring in account access manager ... 1857
Quotas ... 1857
Group assignment quota ... 1858
Throttle limits ... 1858
Code examples ... 1859
IAM ... 1866
Basics ... 1872
Scenarios ... 2661
AWS STS ... 3154
Basics ... 3155
Scenarios ... 3187
Security ... 3439
AWS security credentials ... 3440
Security considerations ... 3441
Programmatic access ... 3442
AWS security audit guidelines ... 3446
When to perform a security audit ... 3447
Guidelines for auditing ... 3447
Review your AWS account credentials ... 3447
Review your IAM users ... 3448
Review your IAM groups ... 3448
Review your IAM roles ... 3449
Review your IAM providers for SAML and OpenID Connect (OIDC) ... 3449
Review Your mobile apps ... 3449
Tips for reviewing IAM policies ... 3450
Data protection ... 3451
Data encryption in IAM and AWS STS ... 3452
Key management in IAM and AWS STS ... 3453
Internetwork traffic privacy in IAM and AWS STS ... 3453
Logging and monitoring ... 3453
Log events with CloudTrail ... 3454
Track privileged tasks in CloudTrail ... 3481
Compliance validation ... 3485
Resilience ... 3485

Best practices for IAM resilience ... 3487
Infrastructure security ... 3488
Configuration and vulnerability analysis ... 3488
AWS managed policies ... 3489
IAMReadOnlyAccess ... 3489
IAMUserChangePassword ... 3490
IAMAccessAnalyzerFullAccess ... 3490
IAMAccessAnalyzerReadOnlyAccess ... 3491
AccessAnalyzerServiceRolePolicy ... 3492
IAMAuditRootUserCredentials ... 3493
IAMCreateRootUserPassword ... 3495
IAMDeleteRootUserCredentials ... 3496
S3UnlockBucketPolicy ... 3498
SQSUnlockQueuePolicy ... 3499
AccountAccessManagerServiceRolePolicy ... 3500
Policy updates ... 3500
Security features outside IAM ... 3506
IAM Access Analyzer ... 3509
Identifying resources shared with an external entity ... 3509
Identifying internal access to business-critical resources ... 3511
Identifying unused access granted to IAM users and roles ... 3512
Validating policies against AWS best practices ... 3513
Validating policies against your specified security standards ... 3513
Generating policies ... 3513
Pricing for IAM Access Analyzer ... 3514
Findings ... 3514
How findings work ... 3516
Getting started with IAM Access Analyzer ... 3520
Findings dashboard ... 3538
Review findings ... 3542
Filter findings ... 3545
Archive findings ... 3552
Resolve findings ... 3553
Error findings ... 3556
Supported resource types ... 3560
Delegated administrator ... 3569

Archive rules ... 3571
Monitoring with EventBridge ... 3573
Security Hub CSPM integration ... 3585
Logging with CloudTrail ... 3593
Filter keys ... 3596
Using service-linked roles ... 3609
Preview access ... 3612
Previewing access in Amazon S3 console ... 3612
Previewing access with IAM Access Analyzer APIs ... 3614
Checks for validating policies ... 3617
How custom policy checks work ... 3618
Examples of reference policies to check for new access ... 3619
Inspect failed custom policy checks ... 3620
Validate with basic policy checks ... 3620
Policy check reference ... 3623
Validate with custom policy checks ... 3765
IAM Access Analyzer policy generation ... 3767
How policy generation works ... 3767
Service and action-level information ... 3768
Things to know ... 3769
Permissions required ... 3770
Generate a policy based on CloudTrail activity (console) ... 3773
Generate a policy using AWS CloudTrail data in another account ... 3776
Generate a policy based on CloudTrail activity (AWS CLI) ... 3780
Generate a policy based on CloudTrail activity (AWS API) ... 3780
IAM Access Analyzer policy generation services ... 3781
IAM Access Analyzer quotas ... 4338
Troubleshoot IAM ... 4341
I can't sign in to my AWS account ... 4341
I lost my access keys ... 4341
Policy variables aren't working ... 4342
Changes that I make are not always immediately visible ... 4342
I am not authorized to perform: iam:DeleteVirtualMFADevice ... 4343
How do I securely create IAM users? ... 4344
Additional resources ... 4345
Access denied error messages ... 4345

I get "access denied" when I make a request to an AWS service 4346
I get "access denied" when I make a request with temporary security credentials 4348
Access denied examples 4349
Access denied error messages with authorization ID (Preview) 4356
Overview 4356
Concepts and definitions 4357
Troubleshoot using the AWS Management Console 4358
Troubleshoot using AWS API or AWS CLI 4359
When you don't receive an authorization ID 4360
Additional considerations 4360
Supported services 4361
Root user issues 4361
IAM policies 4363
Troubleshoot using the visual editor 4364
Troubleshoot with policy summaries 4368
Troubleshoot policy management 4378
Troubleshoot JSON policy documents 4379
Passkeys and FIDO Security Keys 4386
I can't enable my FIDO security key 4386
I can't sign in using my FIDO security key 4387
I lost or broke my FIDO security key 4388
Other issues 4388
IAM roles 4388
I can't assume a role 4388
A new role appeared in my AWS account 4391
I can't edit or delete a role in my AWS account 4391
I'm not authorized to perform: iam:PassRole 4391
Why can't I assume a role with a 12-hour session? (AWS CLI, AWS API) 4392
I receive an error when I try to switch roles in the IAM console 4392
My role has a policy that allows me to perform an action, but I get "access denied" 4393
The service did not create the role's default policy version 4393
There is no use case for a service role in the console 4395
IAM and Amazon EC2 4396
When I try to launch an instance, I don't see the role in the Amazon EC2 console **IAM Role** list 4396
The credentials on my instance are for the wrong role 4397

When I attempt to call the AddRoleToInstanceProfile, I get an AccessDenied error ... 4397
Amazon EC2: When I try to launch an instance with a role, I get an AccessDenied error 4398
I can't access the temporary security credentials on my EC2 instance ... 4398
What do the errors from the info document in the IAM subtree mean? ... 4399
IAM and Amazon S3 ... 4400
How do I grant anonymous access to an Amazon S3 bucket? ... 4400
I'm signed in as an AWS account root user. Why can't I access an Amazon S3 bucket under my account? ... 4401
SAML 2.0 federation ... 4401
Invalid SAML response ... 4402
RoleSessionName is required ... 4403
Not authorized for AssumeRoleWithSAML ... 4403
Invalid RoleSessionName characters ... 4404
Invalid Source Identity characters ... 4404
Invalid response signature ... 4404
Invalid private key ... 4404
Failed to remove private key ... 4405
Failed to remove private key because the Key ID does not match a private key ... 4405
Failed to assume role ... 4405
Could not parse metadata ... 4405
Unable to update identity provider ... 4406
Unable to set assertion encryption mode to Required because no private key is provided 4406
Unable to add and remove private keys in the same request ... 4406
Specified provider doesn't exist ... 4407
DurationSeconds exceeds MaxSessionDuration ... 4407
Private key limit of 2 is reached ... 4407
Response does not contain the required audience ... 4407
How IAM works with other AWS services ... 4408
Create IAM resources with CloudFormation ... 4408
IAM and CloudFormation templates ... 4409
Learn more about CloudFormation ... 4409
Use AWS CloudShell with IAM ... 4410
Obtain IAM permissions for AWS CloudShell ... 4410
Interact with IAM ... 4410
Working with AWS SDKs ... 4412

Reference 4414
Identify AWS resources with Amazon Resource Names (ARNs) 4414
ARN format 4414
Look up the ARN format for a resource 4416
Paths in ARNs 4416
IAM identifiers 4418
Friendly names and paths 4418
IAM ARNs 4419
Unique identifiers 4426
Understanding unique ID prefixes 4428
Getting the unique identifier 4429
IAM and AWS STS quotas 4430
IAM name requirements 4430
IAM object quotas 4431
IAM Access Analyzer quotas 4432
IAM Roles Anywhere quotas 4432
STS request quotas 4432
IAM and STS character limits 4434
Dual-stack endpoint support 4439
IAM dual-stack endpoint support 4439
AWS STS dual-stack endpoint support 4440
Interface VPC endpoints 4441
VPC endpoint availability 4441
Create a VPC endpoint for IAM 4443
Create a VPC endpoint for AWS STS 4444
Create a VPC endpoint for AWS STS OIDC discovery 4445
Control access to AWS STS with VPC endpoint policies 4447
Use VPC condition keys to control federated access 4455
Services that work with IAM 4461
Services that work with IAM 4463
More information 4492
AWS Signature Version 4 4496
How AWS SigV4 works 4497
How AWS SigV4a works 4498
When to sign requests 4498
Why requests are signed 4499

Additional resources ... 4499
SigV4 request elements ... 4500
Authentication methods ... 4503
Create a signed request ... 4509
Request signature examples ... 4522
Troubleshoot SigV4 ... 4524
Policy reference ... 4529
JSON element reference ... 4530
Policy evaluation logic ... 4641
Policy grammar ... 4667
AWS managed policies for job functions ... 4676
Global condition keys ... 4692
IAM condition keys ... 4775
Actions, resources, and condition keys ... 4836
Resources ... 4837
Identities ... 4837
Credentials (passwords, access keys, and MFA devices) ... 4837
Permissions and policies ... 4838
Federation and delegation ... 4838
IAM and other AWS products ... 4839
Using IAM with Amazon EC2 ... 4839
Using IAM with Amazon S3 ... 4839
Using IAM with Amazon RDS ... 4839
Using IAM with Amazon DynamoDB ... 4839
General security practices ... 4840
General resources ... 4840
Making HTTP query requests ... 4842
Endpoints ... 4842
HTTPS required ... 4843
Signing IAM API requests ... 4843
Document history ... 4845

What is IAM?

Follow us on Twitter

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. With IAM, you can manage permissions that control which AWS resources users can access. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources. IAM provides the infrastructure necessary to control authentication and authorization for your AWS accounts.

For instructions on setting up your AWS account and creating identities, see Getting started with IAM.

Identities

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Use IAM to set up other identities in addition to your root user, such as administrators, analysts, and developers, and grant them access to the resources they need to succeed in their tasks.

Access management

After a user is set up in IAM, they use their sign-in credentials to authenticate with AWS. Authentication is provided by matching the sign-in credentials to a principal (an IAM user, AWS STS federated principal, IAM role, or application) trusted by the AWS account. Next, a request is made to grant the principal access to resources. Access is granted in response to an authorization request if the user has been given permission to the resource. For example, when you first sign in to the console and are on the console Home page, you aren't accessing a specific service. When you select a service, the request for authorization is sent to that service and it looks to see if your identity is on the list of authorized users, what policies are being enforced to control the level of access granted, and any other policies that might be in effect. Authorization requests can be made by principals within your AWS account or from another AWS account that you trust.

Once authorized, the principal can take action or perform operations on resources in your AWS account. For example, the principal could launch a new Amazon Elastic Compute Cloud instance, modify IAM group membership, or delete Amazon Simple Storage Service buckets.

Tip

AWS Skill Builder provides a 10-minute video introduction to IAM: Introduction to AWS Identity and Access Management on the AWS Skill Builder website.

Service availability

IAM, like many other AWS services, is eventually consistent. IAM achieves high availability by replicating data across multiple servers within Amazon's data centers around the world. If a request to change some data is successful, the change is committed and safely stored. However, the change must be replicated across IAM, which can take some time. Such changes include creating or updating users, groups, roles, or policies. We recommend that you do not include such IAM changes in the critical, high-availability code paths of your application. Instead, make IAM changes in a separate initialization or setup routine that you run less frequently. Also, be sure to verify that the changes have been propagated before production workflows depend on them. For more information, see Changes that I make are not always immediately visible.

Service cost information

AWS Identity and Access Management (IAM), AWS IAM Identity Center and AWS Security Token Service (AWS STS) are features of your AWS account offered at no additional charge. You are charged only when you access other AWS services using your IAM users or AWS STS temporary security credentials.

IAM Access Analyzer external access analysis is offered at no additional charge. However, you will incur charges for unused access analysis and customer policy checks. For a complete list of charges and prices for IAM Access Analyzer, see IAM Access Analyzer pricing.

For information about the pricing of other AWS products, see the Amazon Web Services pricing page.

Integration with other AWS services

IAM is integrated with many AWS services. For a list of AWS services that work with IAM and the IAM features the services support, see AWS services that work with IAM.

Why should I use IAM?

AWS Identity and Access Management is a powerful tool for securely managing access to your AWS resources. One of the primary benefits of using IAM is the ability to grant shared access to your AWS account. Additionally, IAM allows you to assign granular permissions, enabling you to control exactly what actions different users can perform on specific resources. This level of access control is crucial for maintaining the security of your AWS environment. IAM also provides several other security features. You can add multi-factor authentication (MFA) for an extra layer of protection, and leverage identity federation to seamlessly integrate users from your corporate network or other identity providers. IAM also integrates with AWS CloudTrail, providing detailed logging and identity information to support auditing and compliance requirements. By taking advantage of these capabilities, you can help ensure that access to your critical AWS resources is tightly controlled and secure.

Shared access to your AWS account

You can grant other people permission to administer and use resources in your AWS account without having to share your password or access key.

Granular permissions

You can grant different permissions to different people for different resources. For example, you might allow some users complete access to Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, Amazon Redshift, and other AWS services. For other users, you can allow read-only access to just some Amazon S3 buckets, or permission to administer just some Amazon EC2 instances, or to access your billing information but nothing else.

Secure access to AWS resources for applications that run on Amazon EC2

You can use IAM features to securely provide credentials for applications that run on EC2 instances. These credentials provide permissions for your application to access other AWS resources. Examples include S3 buckets and DynamoDB tables.

Multi-factor authentication (MFA)

You can add two-factor authentication to your account and to individual users for extra security. With MFA you or your users must provide not only a password or access key to work with your

account, but also a code from a specially configured device. If you already use a FIDO security key with other services, and it has an AWS supported configuration, you can use WebAuthn for MFA security. For more information, see Supported configurations for using passkeys and security keys

Identity federation

You can allow users who already have passwords elsewhere—for example, in your corporate network or with an internet identity provider—to access your AWS account. These users are granted temporary credentials that comply with IAM best practice recommendations. Using identity federation enhances the security of your AWS account.

Identity information for assurance

If you use AWS CloudTrail, you receive log records that include information about those who made requests for resources in your account. That information is based on IAM identities.

PCI DSS Compliance

IAM supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with Payment Card Industry (PCI) Data Security Standard (DSS). For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see PCI DSS Level 1.

When do I use IAM?

AWS Identity and Access Management is a core infrastructure service that provides the foundation for access control based on identities within AWS. You use IAM every time you access your AWS account. The way you use IAM will depend on the specific responsibilities and job functions within your organization. Users of AWS services use IAM to access the AWS resources required for their day-to-day work, with administrators granting the appropriate permissions. IAM administrators, on the other hand, are responsible for managing IAM identities and writing policies to control access to resources. Regardless of your role, you interact with IAM whenever you authenticate and authorize access to AWS resources. This could involve signing in as an IAM user, assuming an IAM role, or leveraging identity federation for seamless access. Understanding the various IAM capabilities and use cases is crucial for effectively managing secure access to your AWS environment. When it comes to creating policies and permissions, IAM provides a flexible and granular approach. You can define trust policies to control which principals can assume a role,

in addition to identity-based policies that specify the actions and resources a user or role can access. By configuring these IAM policies, you can help ensure that users and applications have the appropriate level of permissions to perform their required tasks.

When you are performing different job functions

AWS Identity and Access Management is a core infrastructure service that provides the foundation for access control based on identities within AWS. You use IAM every time you access your AWS account.

How you use IAM differs, depending on the work that you do in AWS.

- Service user – If you use an AWS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more advanced features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator.
- Service administrator – If you're in charge of an AWS resource at your company, you probably have full access to IAM. It's your job to determine which IAM features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM.
- IAM administrator – If you're an IAM administrator, you manage IAM identities and write policies to manage access to IAM.

When you are authorized to access AWS resources

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

When you sign-in as an IAM user

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

When you assume an IAM role

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

When you create policies and permissions

You grant permissions to a user by creating a policy, which is a document that lists the actions that a user can perform and the resources those actions can affect. Any actions or resources that are not explicitly allowed are denied by default. Policies can be created and attached to principals (users, groups of users, roles assumed by users, and resources).

You can use these policies with an IAM role:

- **Trust policy** – Defines which principal can assume the role, and under which conditions. A trust policy is a specific type of resource-based policy for IAM roles. A role can have only one trust policy.
- **Identity-based policies (inline and managed)** – These policies define the permissions that the user of the role is able to perform (or is denied from performing), and on which resources.

Use the Example IAM identity-based policies to help you define permissions for your IAM identities. After you find the policy that you need, choose view the policy to view the JSON for the policy. You can use the JSON policy document as a template for your own policies.

Note

If you are using IAM Identity Center to manage your users, you assign permission sets in IAM Identity Center instead of attaching a permissions policy to a principal. When you assign a permission set to a group or user in AWS IAM Identity Center, IAM Identity Center creates corresponding IAM roles in each account, and attaches the policies specified in the permission set to those roles. IAM Identity Center manages the role, and allows the authorized users you've defined to assume the role. If you modify the permission set, IAM Identity Center ensures that the corresponding IAM policies and roles are updated accordingly.

You can also use account access manager — an IAM feature that lets you assign existing IAM roles you create and manage to IAM Identity Center users and groups, giving you access to the full IAM role feature set. You can use it alongside permission sets or on its own.

For more information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

How do I manage IAM?

Managing AWS Identity and Access Management within an AWS environment involves leveraging a variety of tools and interfaces. The most common method is through the AWS Management Console, a web-based interface that allows you to perform a wide range of IAM administrative tasks, from creating users and roles to configuring permissions.

For users more comfortable with command line interfaces, AWS provides two sets of command line tools - the AWS Command Line Interface and the AWS Tools for Windows PowerShell. These allow you to issue IAM-related commands directly from the terminal, often more efficiently than navigating the console. Additionally, AWS CloudShell enables you to run CLI or SDK commands directly from your web browser, using the permissions associated with your console sign-in.

Beyond the console and command line, AWS offers Software Development Kits (SDKs) for various programming languages, enabling you to integrate IAM management functionality directly into your applications. Alternatively, you can access IAM programmatically using the IAM Query API, which allows you to issue HTTPS requests directly to the service. Leveraging these different management approaches provides you with the flexibility to incorporate IAM into your existing workflows and processes.

Use the AWS Management Console

The AWS Management Console is a web application that comprises and refers to a broad collection of service consoles for managing AWS resources. When you first sign in, you see the console home page. The home page provides access to each service console and offers a single place to access the information for performing your AWS related tasks. Which services and applications are available to you after signing in to the console depend on which AWS resources you have permission to access. You can be granted permissions to resources either through assuming a role, being a member of a group that has been granted permissions, or being explicitly granted permission. For a stand-alone AWS account, the root user or IAM administrator configures access to resources. For AWS Organizations, the management account or delegated administrator configures access to resources.

If you plan to have people using the AWS Management Console to manage AWS resources, we recommend configuring users with temporary credentials as a security best practice. IAM users that have assumed a role, federated principals, and users in IAM Identity Center have temporary credentials, while the IAM user and root user have long-term credentials. Root user credentials provide full access to the AWS account, while other users have credentials that provide access to the resources granted them by IAM policies.

The sign-in experience is different for the different types of AWS Management Console users.

- IAM users and the root user sign-in from the main AWS sign-in URL (https://signin.aws.amazon.com). Once they sign in they have access to the resources in the account to which they have been granted permission.

  To sign in as the root user you must have the root user email address and password.

  To sign in as an IAM user you must have the AWS account number or alias, the IAM user name, and the IAM user password.

  We recommend that you restrict IAM users in your account to specific situations that require long-term credentials, such as for emergency access, and that you use the root user only for tasks that require root user credentials.

  For convenience, the AWS sign-in page uses a browser cookie to remember the IAM user name and account information. The next time the user goes to any page in the AWS Management Console, the console uses the cookie to redirect the user to the account sign-in page.

  Sign out of the console when you finish your session to prevent reuse of your previous sign in.

- IAM Identity Center users sign in using a specific AWS access portal that's unique to their organization. Once they sign in they can choose which account or application to access. If they choose to access an account, they choose which permission set they want to use for the management session.
- OIDC and SAML federated principals managed in an external identity provider linked to an AWS account sign-in using a custom enterprise access portal. The AWS resources available to users are dependent upon the policies selected by their organization.

> **Note**
>
> To provide an additional level of security, root user, IAM users, and users in IAM Identity Center can have multi-factor authentication (MFA) verified by AWS before granting access to AWS resources. When MFA is enabled, you must also have access to the MFA device to sign in.

To learn more about how different users sign-in to the management console, see Sign in to the AWS Management Console in the *AWS Sign-In User Guide*.

AWS Command Line Tools

You can use the AWS command line tools to issue commands at your system's command line to perform IAM and AWS tasks. Using the command line can be faster and more convenient than the console. The command line tools are also useful if you want to build scripts that perform AWS tasks.

AWS provides two sets of command line tools: the AWS Command Line Interface (AWS CLI) and the AWS Tools for Windows PowerShell. For information about installing and using the AWS CLI, see the AWS Command Line Interface User Guide. For information about installing and using the Tools for Windows PowerShell, see the AWS Tools for PowerShell User Guide.

After signing in to the console, you can use AWS CloudShell from your browser to run CLI or SDK commands. The permissions for accessing AWS resources are based on the credentials you used to sign-in to the console. Depending on your experience, you may find the CLI to be a more efficient method of managing your AWS account. For more information, see Use AWS CloudShell to work with AWS Identity and Access Management

AWS Command Line Interface (CLI) and Software Development Kits (SDKs)

IAM Identity Center and IAM users use different methods to authenticate their credentials when they authenticate through the CLI or the application interfaces (APIs) in the associated SDKs.

Credentials and configuration settings are located in multiple places, such as the system or user environment variables, local AWS configuration files, or explicitly declared on the command line as a parameter. Certain locations take precedence over others.

Both IAM Identity Center and IAM provide access keys that can be used with the CLI or SDK. IAM Identity Center access keys are temporary credentials that can be automatically refreshed and are recommended over the long-term access keys associated with IAM users.

To manage your AWS account using the CLI or SDK you can use AWS CloudShell from your browser. If you use CloudShell to run CLI or SDK commands you must first sign-in to the console. The permissions for accessing AWS resources are based on the credentials you used to sign-in to the console. Depending on your experience, you may find the CLI to be a more efficient method of managing your AWS account.

For application development, you can download the CLI or SDK to your computer and sign-in from the command prompt or a Docker window. In this scenario, you configure authentication and access credentials as part of the CLI script or SDK application. You can configure programmatic access to resources in different ways, depending on the environment and the access available to you.

- Recommended options for authenticating local code with AWS service are IAM Identity Center and IAM Roles Anywhere
- Recommended options for authenticating code running within an AWS environment are to use IAM roles or use IAM Identity Center credentials.

When signing in using the AWS access portal, you can get short-term credentials from the start page where you choose your permission set. These credentials have a defined duration and don't automatically refresh. If you want to use these credentials, after signing in to the AWS portal, choose the AWS account and then choose the permissions set. Select **Command line or programmatic access** to view the options you can use to access AWS resources programmatically or from the CLI. For more information about these methods, see Getting and refreshing temporary credentials in the *IAM Identity Center User Guide*. These credentials are often used during application development to quickly test code.

We recommend using IAM Identity Center credentials that automatically refresh when automating access to your AWS resources. If you have configured users and permission sets in IAM Identity Center you use the `aws configure sso` command to use a command-line wizard that will help you identify the credentials available to you and store them in a profile. For more information about configuring your profile, see Configure your profile with the `aws configure sso` wizard in the *AWS Command Line Interface User Guide for Version 2*.

> **Note**
>
> Many sample applications use long-term access keys associated with IAM users or root user. You should only use long-term credentials within a sandbox environment as part of a learning exercise. Review the alternatives to long-term access keys and plan to transition your code to use alternative credentials, such as IAM Identity Center credentials or IAM roles, as soon as possible. After transitioning your code, delete the access keys.

To learn more about configuring the CLI, see Install or update the latest version of the AWS CLI in the *AWS Command Line Interface User Guide for Version 2* and Authentication and access credentials in the *AWS Command Line Interface User Guide*

To learn more about configuring the SDK, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide* and IAM Roles Anywhere in the *AWS SDKs and Tools Reference Guide*.

Use the AWS SDKs

AWS provides SDKs (software development kits) that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests, managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see the Tools for Amazon Web Services page.

Use the IAM Query API

You can access IAM and AWS programmatically by using the IAM Query API, which lets you issue HTTPS requests directly to the service. When you use the Query API, you must include code to digitally sign requests using your credentials. For more information, see Calling the IAM API using HTTP query requests and the IAM API Reference.

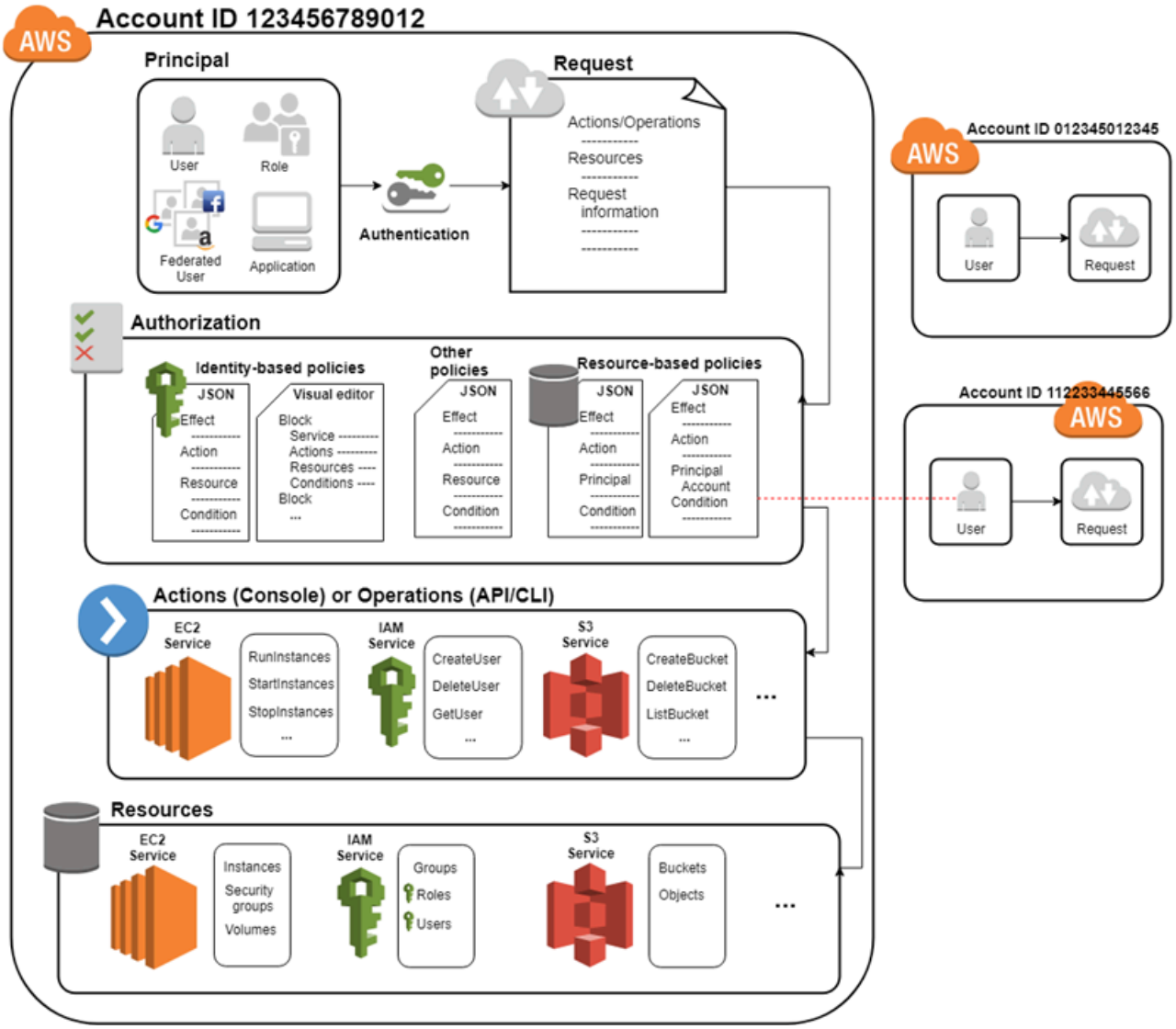
How IAM works

AWS Identity and Access Management provides the infrastructure necessary to control authentication and authorization for your AWS account.

First, a human user or an application uses their sign-in credentials to authenticate with AWS. IAM matches the sign-in credentials to a principal (an IAM user, AWS STS federated user principal, IAM role, or application) trusted by the AWS account and authenticates the principal to access AWS.

Next, IAM makes a request to grant the principal access to resources. IAM grants or denies access in response to an authorization request. For example, when you first sign in to the console and are on the console Home page, you aren't accessing a specific service. When you select a service, you send an authorization request to IAM for that service. IAM verifies that your identity is on the list of authorized users, determines what policies control the level of access granted, and evaluates any other policies that might be in effect. Principals within your AWS account or from another AWS account that you trust can make authorization requests.

Once authorized, the principal can perform actions or operations on resources in your AWS account. For example, the principal could launch a new Amazon Elastic Compute Cloud instance, modify IAM group membership, or delete Amazon Simple Storage Service buckets. The following diagram illustrates this process through the IAM infrastructure:



Components of a request

When a principal tries to use the AWS Management Console, the AWS API, or the AWS CLI, that principal sends a *request* to AWS. The request includes the following information:

- **Actions or operations** – The actions or operations that the principal wants to perform, such as an action in the AWS Management Console, or an operation in the AWS CLI or AWS API.
- **Resources** – The AWS resource object upon which the principal requests to perform an action or operation.
- **Principal** – The person or application that used an entity (user or role) to send the request. Information about the principal includes the permission policies.

- **Environment data** – Information about the IP address, user agent, SSL enabled status, and the timestamp.
- **Resource data** – Data related to the resource requested, such as a DynamoDB table name or a tag on an Amazon EC2 instance.

AWS gathers the request information into a *request context*, which IAM evaluates to authorize the request.

How principals are authenticated

A principal signs in to AWS using their credentials which IAM authenticates to permit the principal to send a request to AWS. Some services, such as Amazon S3 and AWS STS, allow specific requests from anonymous users. However, they're the exception to the rule. Each type of user goes through authentication.

- **Root user** – Your sign in credentials used for authentication are the email address you used to create the AWS account and the password you specified at that time.
- **Federated principal** – Your identity provider authenticates you and passes your credentials to AWS, you don't have to sign-in directly to AWS. Both IAM Identity Center and IAM support identity federation.
- **Users in AWS IAM Identity Center directory***(not federated)*– Users created directly in the IAM Identity Center default directory sign in using the AWS access portal and provide your username and password.
- **IAM user** – You sign-in by providing your account ID or alias, your username, and password. To authenticate workloads from the API or AWS CLI, you might use temporary credentials through assuming a role or you might use long-term credentials by providing your access key and secret key.

  To learn more about the IAM entities, see IAM users and IAM roles.

AWS recommends that you use multi-factor authentication (MFA) with all users to increase the security of your account. To learn more about MFA, see AWS Multi-factor authentication in IAM.

Authorization and permission policy basics

Authorization refers to the principal having the required permissions to complete their request. During authorization, IAM identifies the policies that apply to the request using values from the

request context. It then uses the policies to determine whether to allow or deny the request. IAM stores most permission policies as JSON documents that specify the permissions for principal entities.

There are several types of policies that can affect an authorization request. To provide your users with permissions to access the AWS resources in your account, you can use identity-based policies. Resource-based policies can grant cross-account access. To make a request in a different account, a policy in the other account must allow you to access the resource *and* the IAM entity that you use to make the request must have an identity-based policy that allows the request.

IAM checks each policy that applies to the context of your request. IAM policy evaluation uses an *explicit deny*, which means that if a single permissions policy includes a denied action, IAM denies the entire request and stops evaluating. Because requests are *denied by default*, the applicable permissions policies must allow every part of your request for IAM to authorize your request. The evaluation logic for a request within a single account follows these basic rules:

- By default, all requests are denied. (In general, requests made using the AWS account root user credentials for resources in the account are always allowed.)
- An explicit allow in any permissions policy (identity-based or resource-based) overrides this default.
- The existence of an AWS Organizations service control policy (SCP) or resource control policy (RCP), IAM permissions boundary, or a session policy overrides the allow. If one or more of these policy types exists, they must all allow the request. Otherwise, it's implicitly denied. For more information on SCPs and RCPs, see Authorization policies in AWS Organizations in the *AWS Organizations User Guide*.
- An explicit deny in any policy overrides any allows in any policy.

To learn more, see Policy evaluation logic.

After IAM authenticates and authorizes the principal, IAM approves the actions or operations in their request by evaluating the permission policy that applies to the principal. Each AWS service defines the actions (operations) they support, and include things that you can do to a resource, such as viewing, creating, editing, and deleting that resource. The permission policy that applies to the principal must include the necessary actions to perform an operation. To learn more about how IAM evaluates permission policies, see the section called “Policy evaluation logic”.

The service defines a set of actions that a principal can perform on each resource. When creating permission policies, make sure to include the actions that you want the user to be able to perform.

For example, IAM supports over 40 actions for a user resource, including the following basic actions:

- `CreateUser`
- `DeleteUser`
- `GetUser`
- `UpdateUser`

In addition, you can specify conditions in your permission policy that provide access to resources when the request meets the specified conditions. For example, you might want a policy statement to take effect after a specific date or to control access when a specific value appears in an API. To specify conditions, you use the `Condition` element of a policy statement.

After IAM approves the operations in a request, the principal can work with the related resources in your account. A resource is an object that exists within a service. Examples include an Amazon EC2 instance, an IAM user, and an Amazon S3 bucket. If the principal creates a request to perform an action on a resource that isn't included in the permission policy, the service denies the request. For example, if you have permission to delete an IAM role but request to delete an IAM group, the request fails if you don't have permission to delete IAM groups. To learn more about which actions, resources, and condition keys the different AWS services support, see Actions, Resources, and Condition Keys for AWS Services.

Compare IAM identities and credentials

The identities managed in AWS Identity and Access Management are IAM users, IAM roles, and IAM groups. These identities are in addition to your root user that AWS created along with your AWS account.

We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, provision additional users and grant them the permissions required to perform the necessary tasks. You can add users either by adding people to your IAM Identity Center directory, federating an external identity provider with either IAM Identity Center or IAM or creating least privilege IAM users.

For additional security, we recommend centralizing root access to help you centrally secure the root user credentials of your AWS accounts managed using AWS Organizations. Centrally manage root access for member accounts lets you centrally remove and prevent long-term root user credential

recovery, preventing unintended root access at scale. After you enable centralized root access, you can assume a privileged session to perform actions on member accounts.

After you have set up your users, you can give access to your AWS account to specific people and provide them permissions to access resources.

As a best practice, AWS recommends that you require human users to assume an IAM role to access AWS so that they're using temporary credentials. If you are managing identities in the IAM Identity Center directory or using federation with an identity provider you are following best practices.

Terms

These terms are commonly used when working with IAM identities:

IAM Resource

The IAM service stores these resources. You can add, edit, and remove them from the Console.

- IAM user
- IAM group
- IAM role
- Permission policy
- Identity-provider object

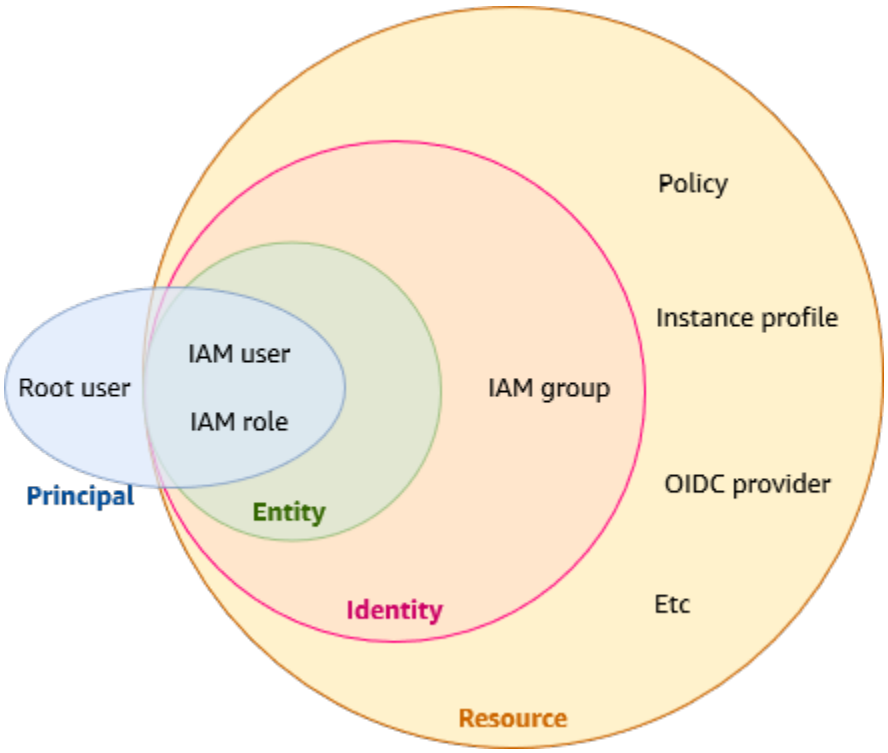
IAM Entity

IAM resources that AWS uses for authentication. Specify the entity as a Principal in a resource-based policy.

- IAM user
- IAM role

IAM Identity

The IAM resource that's authorized in policies to perform actions and to access resources. Identities include IAM users, IAM groups, and IAM roles.



Principals

An AWS account root user, IAM user or an IAM role that can make a request for an action or operation on an AWS resource. Principals include human users, workloads, federated principals, and assumed roles. After authentication, IAM grants the principal either permanent or temporary credentials to make requests to AWS, depending on the principal type.

Human users are also known as *human identities*, such as the people, administrators, developers, operators, and consumers of your applications.

Workloads are a collection of resources and code that delivers business value, such as an application, process, operational tools, and other components.

Federated principals are users whose identity and credentials are managed by another identity provider, such as Active Directory, Okta, or Microsoft Entra.

IAM roles are an IAM identity that you can create in your account that has specific permissions that determine what the identity can and can't do. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it.

IAM grants IAM users and the root user long-term credentials and IAM roles temporary credentials. users in AWS IAM Identity Center, OIDC and SAML federated principals assume IAM

roles when they sign-in to AWS, which grants them temporary credentials. As a best practice, we recommend that you require human users and workloads to access AWS resources using temporary credentials.

Difference between IAM users and users in IAM Identity Center

IAM users aren't separate accounts; they're individual users within your account. Each user has their own password for access to the AWS Management Console. You can also create an individual access key for each user so that the user can make programmatic requests to work with resources in your account.

IAM users and their access keys have long-term credentials to your AWS resources. The primary use for IAM users is to give workloads that can't use IAM roles the ability to make programmatic requests to AWS services using the API or CLI.

> **Note**
>
> For scenarios in which you need IAM users with programmatic access and long-term credentials, we recommend that you update access keys when needed. For more information, see Update access keys.

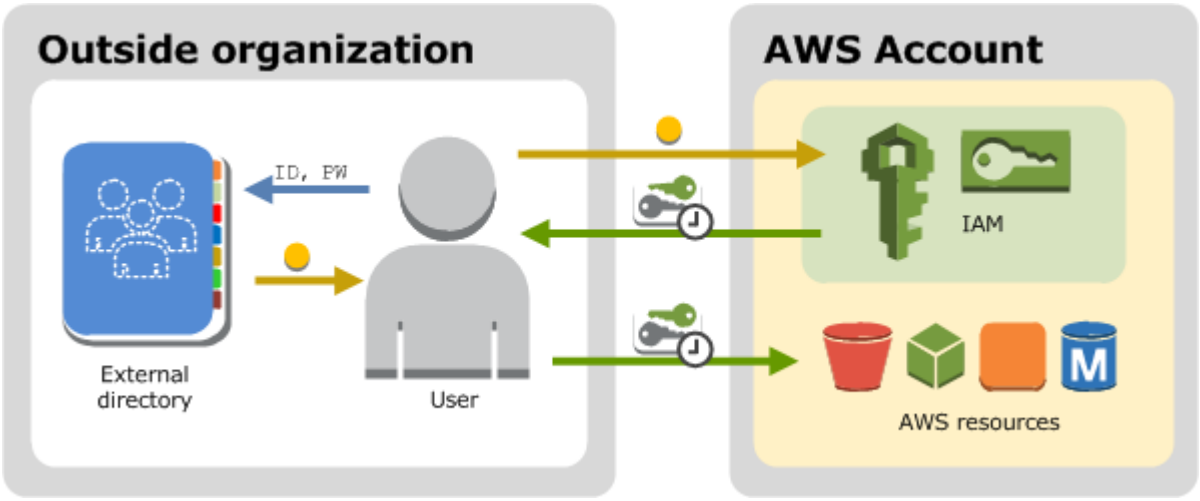
Workforce identities (people) are **users in AWS IAM Identity Center** that have different permission needs depending on the role they're performing and can work in various AWS accounts across an organization. If you have use cases that require access keys, you can support those use cases with users in AWS IAM Identity Center. People who sign-in through the AWS access portal can obtain access keys with short-term credentials to your AWS resources. For centralized access management, we recommend that you use AWS IAM Identity Center (IAM Identity Center) to manage access to your accounts and permissions within those accounts. IAM Identity Center is automatically configured with an Identity Center directory as your default identity source where you can add people and groups, and assign their level of access to your AWS resources. For more information, see What is AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

The main difference between these two types of users is that users in IAM Identity Center automatically assume an IAM role when they sign-in to AWS before they access the management console or AWS resources. IAM roles grant temporary credentials each time the user signs-in to AWS. For IAM users to sign in using an IAM role they must have permission to assume and switch

roles and they must explicitly choose to switch to the role they want to assume after accessing the AWS account.

Federate users from an existing identity source

If the users in your organization are already authenticated when they sign in to your corporate network, you don't have to create separate IAM users or users in IAM Identity Center for them. Instead, you can *federate* those user identities into AWS using either IAM or AWS IAM Identity Center. OIDC and SAML federated principals assume an IAM role that gives them permissions to access specific resources. For more information about roles, see Roles terms and concepts.



Federation is useful in these cases:

- **Your users already exist in a corporate directory.**

  If your corporate directory is compatible with Security Assertion Markup Language 2.0 (SAML 2.0), you can configure your corporate directory to provide single-sign on (SSO) access to the AWS Management Console for your users. For more information, see Common scenarios for temporary credentials.

  If your corporate directory isn't compatible with SAML 2.0, you can create an identity broker application to provide single-sign on (SSO) access to the AWS Management Console for your users. For more information, see Enable custom identity broker access to the AWS console.

  If your corporate directory is Microsoft Active Directory, you can use AWS IAM Identity Center to connect a self-managed directory in Active Directory or a directory in AWS Directory Service to establish trust between your corporate directory and your AWS account.

If you are using an external identity provider (IdP) such as Okta or Microsoft Entra to manage users, you can use AWS IAM Identity Center to establish trust between your IdP and your AWS account. For more information, see Connect to an external identity provider in the *AWS IAM Identity Center User Guide*.

- **Your users already have Internet identities.**

  If you are creating a mobile app or web-based app that can let users identify themselves through an Internet identity provider like Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC) compatible identity provider, the app can use federation to access AWS. For more information, see OIDC federation.

> **Tip**
>
> To use identity federation with Internet identity providers, we recommend you use Amazon Cognito.

Different methods to provide user access

Here are the ways you can provide access to your AWS resources.

Type of user access	When is it used?	Where is more information?
Single sign-on access for people, such as your workforce users, to AWS resources using IAM Identity Center	IAM Identity Center provides a central place that brings together administration of users and their access to AWS accounts and cloud applicati ons. You can set up an identity store within IAM Identity Center or you can configure federation with an existing identity provider (IdP). Security best practices	For more information about setting up IAM Identity Center, see Getting Started in the *AWS IAM Identity Center User Guide* For more information about using MFA in IAM Identity Center, see Multi-factor authentic ation in the *AWS IAM Identity Center User Guide*

Type of user access	When is it used?	Where is more information?
	recommend granting your human users limited credentials to AWS resources. People have an easier sign-in experience and you maintain control over their access to resources from a single system. IAM Identity Center supports multi-factor authentication (MFA) for additional account security.	
Federated access for human users, such as your workforce users, to AWS services using IAM identity providers (IdPs)	IAM supports IdPs that are compatible with OpenID Connect (OIDC) or SAML 2.0 (Security Assertion Markup Language 2.0). After you create an IAM identity provider, create one or more IAM roles that can be dynamically assigned to a federated principal.	For more information about IAM identity providers and federation, see Identity providers and federation into AWS.

Type of user access	When is it used?	Where is more information?
Cross-account access between AWS accounts	You want to share access to certain AWS resources with users in other AWS accounts. Roles are the primary way to grant cross-account access. However, some AWS services support resource-based policies that allow you to attach a policy directly to a resource (instead of using a role as a proxy).	For more information about IAM roles, see IAM roles. For more information about service-linked roles, see Create a service-linked role. For information about which services support using service-linked roles, see AWS services that work with IAM. Find the services that have **Yes** in the **Service-Linked Role** column. To view the service-linked role documentation for that service select the link associated with the **Yes** in that column.

Type of user access	When is it used?	Where is more information?
Long-term credentials for designated IAM users in your AWS account	You might have specific use cases that require long-term credentials with IAM users in AWS. You can use IAM to create these IAM users in your AWS account, and use IAM to manage their permissions. Some use cases include the following: • Workloads that can't use IAM roles • Third-party AWS clients that require programmatic access through access keys • Service-specific credentials for AWS CodeCommit or Amazon Keyspaces • AWS IAM Identity Center isn't available for your account and you have no other identity provider As a best practice in scenarios in which you need IAM users with programmatic access and long-term credentials, we recommend that you update access keys when needed. For more information, see Update access keys.	For more information about setting up an IAM user, see Create an IAM user in your AWS account. For more information about IAM user access keys, see Manage access keys for IAM users. For more information about service-specific credentials for AWS CodeCommit or Amazon Keyspaces, see IAM credentials for CodeCommit: Git credentials, SSH keys, and AWS access keys and Use IAM with Amazon Keyspaces (for Apache Cassandra).

Support programmatic user access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS:

- If you manage identities in IAM Identity Center, the AWS APIs require a profile, and the AWS Command Line Interface requires a profile or an environment variable.
- If you have IAM users, the AWS APIs and the AWS Command Line Interface require access keys. Whenever possible, create temporary credentials that consist of an access key ID, a secret access key, and a security token that indicates when the credentials expire.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	Option	More information
Workforce identities (People and users managed in IAM Identity Center)	Use short-term credentials to sign programmatic requests to the AWS CLI or AWS APIs (directly or by using the AWS SDKs).	For the AWS CLI, follow the instructions in Getting IAM role credentials for CLI access in the *AWS IAM Identity Center User Guide*. For the AWS APIs, follow the instructions in SSO credentials in the *AWS SDKs and Tools Reference Guide*.
IAM users	Use short-term credentials to sign programmatic requests to the AWS CLI or AWS APIs (directly or by using the AWS SDKs).	Follow the instructions in Using temporary credentials with AWS resources.
IAM users	Use long-term credentials to sign programmatic requests to the AWS CLI or AWS APIs	Follow the instructions in Managing access keys for IAM users.

Which user needs programmatic access?	Option	More information
	(directly or by using the AWS SDKs). (Not recommended)	
Federated principals	Use an AWS STS API operation to create a new session with temporary security credentials that include an access key pair and a session token.	For explanations of the API operations, see the section called “Request temporary security credentials”

How permissions and policies provide access management

The access management portion of AWS Identity and Access Management (IAM) helps you define what a principal entity can do in an account. A principal entity is a person or application authenticated using an IAM entity (IAM user or IAM role). Access management is often referred to as *authorization*. You manage access in AWS by creating policies and attaching them to IAM identities (IAM users, IAM groups, or IAM roles) or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal uses an IAM entity (IAM user or IAM role) to make a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about policy types and uses, see Policies and permissions in AWS Identity and Access Management.

Policies and accounts

If you manage a single account in AWS, then you define the permissions within that account using policies. If you manage permissions across multiple accounts, it's more difficult to manage permissions for your IAM users. You can use IAM roles, resource-based policies, or access control lists (ACLs) for cross-account permissions. However, if you own multiple accounts, we instead recommend using the AWS Organizations service to help you manage those permissions. For more information, see What is AWS Organizations? in the *AWS Organizations User Guide*.

Policies and users

IAM users are identities in the AWS account. When you create an IAM user, they can't access anything in your account until you give them permission. You give permissions to an IAM user by creating an identity-based policy, which is a policy attached to the IAM user or an IAM group to which the IAM user belongs. The following example shows a JSON policy that permits the IAM user to perform all Amazon DynamoDB actions (`dynamodb:*`) on the `Books` table in the `123456789012` account within the `us-east-2` Region.

JSON

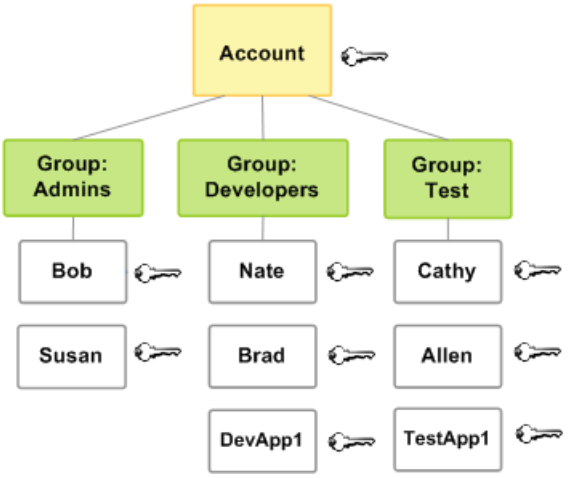
```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "dynamodb:*",
    "Resource": "arn:aws:dynamodb:us-east-2:123456789012:table/Books"
  }
}
```

After you attach this policy to your IAM user, the user has permission to perform all actions in the `Books` table of your DynamoDB instance. Most IAM users have multiple policies that combine to represent their total granted permissions.

Actions or resources that aren't explicitly allowed by a policy are denied by default. For example, if the preceding policy is the single policy attached to a user, then that user can perform DynamoDB actions on the `Books` table, but can't perform actions on other tables. Similarly, the user isn't allowed to perform any actions in Amazon EC2, Amazon S3, or in any other AWS service because permissions to work with those services aren't included in the policy.

Policies and IAM groups

You can organize IAM users into *IAM groups* and attach a policy to the IAM group. In that case, individual IAM users still have their own credentials, but all the IAM users in the IAM group have the permissions attached to the IAM group. Use IAM groups for easier permissions management.



IAM users or IAM groups can have multiple policies attached to them that grant different permissions. In that case, the combination of policies determines the effective permissions for the principal. If the principal doesn't have explicit `Allow` permission for both an action and a resource, the principal doesn't have those permissions.

Federated user sessions and roles

Federated principals don't have permanent identities in your AWS account the way that IAM users do. To assign permissions to federated principals, you can create an entity referred to as a *role* and define permissions for the role. When a SAML or OIDC federated principal signs in to AWS, the user is associated with the role and is granted the permissions that are defined in the role. For more information, see Create a role for a third-party identity provider.

Identity-based and resource-based policies

Identity-based policies are permissions policies that you attach to an IAM identity, such as an IAM user, group, or role. Resource-based policies are permissions policies that you attach to a resource such as an Amazon S3 bucket or an IAM role trust policy.

Identity-based policies control what actions the identity can perform, on which resources, and under what conditions. Identity-based policies can be further categorized:

- **Managed policies** – Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. You can use two types of managed policies:
  - **AWS managed policies** – Managed policies that are created and managed by AWS. If you are new to using policies, we recommend that you start by using AWS managed policies.

- **Customer managed policies** – Managed policies that you create and manage in your AWS account. Customer managed policies provide more precise control over your policies than AWS managed policies. You can create, edit, and validate an IAM policy in the visual editor or by creating the JSON policy document directly. For more information, see Define custom IAM permissions with customer managed policies and Edit IAM policies.
- **Inline policies** – Policies that you create and manage and that are embedded directly into a single user, group, or role. In most cases, we don't recommend using inline policies.

Resource-based policies control what actions a specified principal can perform on that resource and under what conditions. Resource-based policies are inline policies, and there are no managed resource-based policies. To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy.

The IAM service supports one type of resource-based policy called a role *trust policy*, which you attach to an IAM role. Because an IAM role is both an identity and a resource that supports resource-based policies, you have to attach both a trust policy and an identity-based policy to an IAM role. Trust policies define which principal entities (accounts, users, roles, and AWS STS federated user principals) can assume the role. To learn how IAM roles are different from other resource-based policies, see Cross account resource access in IAM.

To see which services support resource-based policies, see AWS services that work with IAM. To learn more about resource-based policies, see Identity-based policies and resource-based policies.

Define permissions based on attributes with ABAC authorization

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. AWS calls these attributes *tags*. You can attach tags to IAM resources, including IAM entities (IAM users or IAM roles) and to AWS resources. You can create a single ABAC policy or small set of policies for your IAM principals. You can design ABAC policies that allow operations when the principal's tag matches the resource tag. ABAC's attribute system that provides both high user context and granular access control. Because ABAC is attribute-based, it can perform dynamic authorization for data or applications that grants or revokes access in real time. ABAC is helpful in environments that are scaling and in situations where identity or resource policy management has become complex.

For example, you can create three IAM roles with the `access-project` tag key. Set the tag value of the first IAM role to `Heart`, the second to `Star`, and the third to `Lightning`. You can then use a single policy that allows access when the IAM role and the AWS resource have the tag value `access-project`. For a detailed tutorial that demonstrates how to use ABAC in AWS, see IAM tutorial: Define permissions to access AWS resources based on tags. To learn about services that support ABAC, see AWS services that work with IAM.
This diagram illustrates that the tags applied to a principal must match the tags applied to a resource for the user to be granted permissions to the resource. Tags be applied to IAM groups, resource groups, individual users and individual resources.

Comparison of ABAC to the traditional RBAC model

The traditional authorization model used in IAM is role-based access control (RBAC). RBAC defines permissions based on a person's job function, or *role*, which is distinct from an IAM role. IAM does include managed policies for job functions that align permissions to a job function in an RBAC model.

In IAM, you implement RBAC by creating different policies for different job functions. You then attach the policies to identities (IAM users, IAM groups, or IAM roles). As a best practice, you grant the minimum permissions necessary for the job function. This results in least privilege access. Each job function policy lists the specific resources that identities assigned that policy can access. The disadvantage to using the traditional RBAC model is that when you or your users add new resources to your environment, you have to update the policies to allow access to those resources.

For example, assume that you have three projects, named `Heart`, `Star`, and `Lightning`, on which your employees work. You create an IAM role for each project. You then attach policies to each IAM role to define the resources that anyone allowed to assume the IAM role can access. If an employee changes jobs within your company, you assign them to a different IAM role. You can assign people or programs to more than one IAM role. However, the `Star` project might require additional resources, such as a new Amazon EC2 container. In that case, you have to update the policy attached to the `Star` IAM role to specify the new container resource. Otherwise, `Star` project members aren't allowed to access the new container.
This diagram illustrates that role-based access control requires that each identity be assigned a specific job-function based policy to access different resources.

ABAC provides the following advantages over the traditional RBAC model:

- **ABAC permissions scale with innovation.** It's no longer necessary for an administrator to update existing policies to allow access to new resources. For example, assume that you designed

your ABAC strategy with the `access-project` tag. A developer uses the IAM role with the `access-project` = `Heart` tag. When people on the `Heart` project need additional Amazon EC2 resources, the developer can create new Amazon EC2 instances with the `access-project` = `Heart` tag. Then anyone on the `Heart` project can start and stop those instances because their tag values match.

- **ABAC requires fewer policies.** Because you don't have to create different policies for different job functions, you create fewer policies. Those policies are easier to manage.
- **Using ABAC, teams can dynamically respond to change and growth.** Because permissions for new resources are automatically granted based on attributes you don't have to manually assign policies to identities. For example, if your company already supports the `Heart` and `Star` projects using ABAC, it's easy to add a new `Lightning` project. An IAM administrator creates a new IAM role with the `access-project` = `Lightning` tag. It's not necessary to change the policy to support a new project. Anyone that has permissions to assume the IAM role can create and view instances tagged with `access-project` = `Lightning`. Another scenario is when a team member moves from the `Heart` project to the `Lightning` project. To provide team member access to the `Lightning` project , the IAM administrator assigns them to a different IAM role. It's not necessary to change the permissions policies.
- **Granular permissions are possible using ABAC.** When you create policies, it's a best practice to grant least privilege. Using traditional RBAC, you write a policy that allows access to specific resources. However, when you use ABAC, you can allow actions on all resources if the resource's tag matches the principal's tag.
- **Use employee attributes from your corporate directory with ABAC.** You can configure your SAML or OIDC provider to pass session tags to IAM. When your employees federate into AWS, IAM applies their attributes to their resulting principal. You can then use ABAC to allow or deny permissions based on those attributes.

For a detailed tutorial that demonstrates how to use ABAC in AWS, see IAM tutorial: Define permissions to access AWS resources based on tags.

Getting started with IAM

AWS Identity and Access Management (IAM) helps you securely control access to Amazon Web Services (AWS) and your account resources. IAM can also keep your sign-in credentials private. You don't specifically sign up to use IAM. There is no charge to use IAM.

How you use IAM depends on your job function. See When do I use IAM? for role-specific guidance.

Use IAM to give identities, such as users and roles, access to resources in your account. For example, you can use IAM with existing users in your corporate directory that you manage external to AWS or you can create users in AWS using AWS IAM Identity Center. Federated identities assume defined IAM roles to access the resources they need. For more information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide.*

> **Note**
>
> IAM is integrated with several AWS products. For a list of services that support IAM, see AWS services that work with IAM.

To learn about getting started with AWS, creating an administrative user, an AWS Organizations, and using multiple services to solve a problem such as building and launching your first project, see the Getting Started Resource Center.

For hands-on practice with common IAM tasks, see IAM tutorials. Tasks include creating roles for cross-account access and using tags for attribute-based access control.

After you set up your account, plan how people and workloads access it. For more information, see Plan access to your AWS account.

Then use Prepare for least-privilege permissions to follow the principle of least privilege.

Setting up your AWS account

Before you start working with IAM, make sure you have completed the initial set up of your AWS environment.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

When you signed up for the service, you created an AWS account using an email address and a password. Those are your AWS root user credentials. As a best practice, you don't use your root user credentials to access AWS for daily tasks. Only use your root user credentials to perform tasks that require root user credentials. Also, do not share your credentials with anyone else. Instead, add people to your directory and give them access to your AWS account.

To secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Grant access to the billing console

IAM users and roles in an AWS account can't access the Billing and Cost Management console by default. This is true even if they have IAM policies that grant access to certain Billing features. To grant access, the AWS account root user must first activate IAM access.

> **Note**
>
> As a security best practice, we recommend that you provide access to your resources through identity federation with AWS IAM Identity Center. When you enable IAM Identity Center in conjunction with AWS Organizations, the Billing and Cost Management console is enabled by default with consolidated billing for all AWS accounts in your organization. For more information, see Consolidating billing for AWS Organizations in the *Billing and Cost Management User Guide*.

1. Sign in to the AWS Management Console with your root user credentials (specifically, the email address and password that you used to create your AWS account).
2. On the navigation bar, select your account name, and then select Account.
3. Scroll down the page until you find the section **IAM User and Role Access to Billing Information**, then select **Edit**.
4. Select the **Activate IAM Access** check box to activate access to the Billing and Cost Management console pages.
5. Choose **Update**.

   The page displays the message **IAM user/role access to billing information is activated**.

   > **Important**
   >
   > Activating IAM access alone doesn't grant any permissions for users or roles to view the Billing and Cost Management console pages. You must also attach the required identity-based policies to IAM roles to grant access to the billing console. Roles provide temporary credentials that users can assume when needed.

6. Use the AWS Management Console to create a role that a user can assume to access the billing console.
7. On the **Add permissions** page for the role, add permissions to list and view details about the Billing resources in your AWS account.

   The AWS managed policy Billing grants users permission to view and edit the Billing and Cost Management console. This includes viewing account usage, modifying budgets and payment methods. For more policy examples that you can attach to IAM roles to control access to your account's billing information, see AWS Billing policy examples in the *Billing and Cost Management User Guide*.

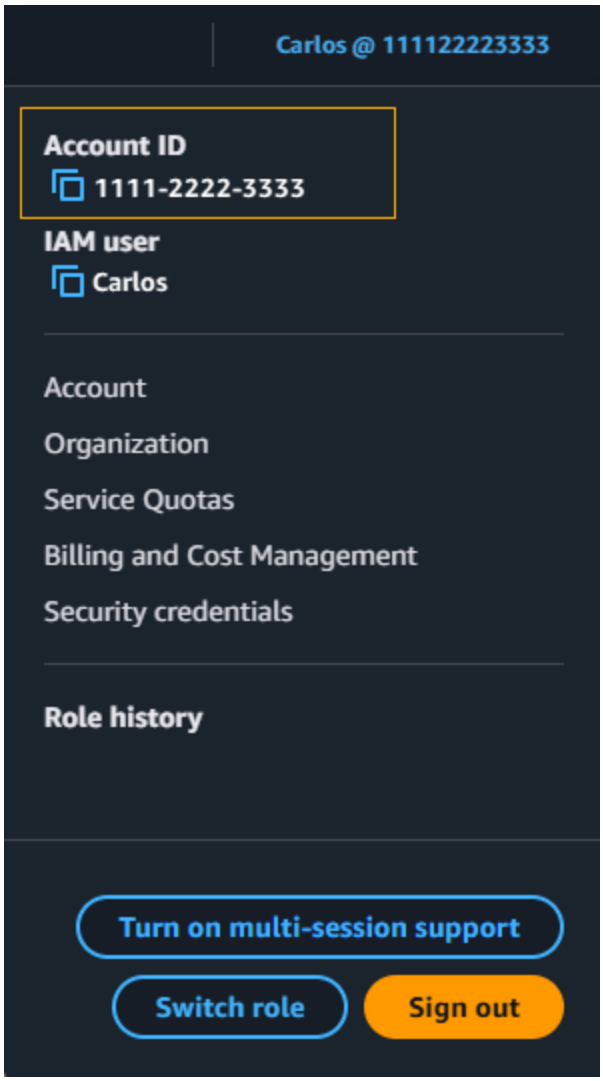
Viewing your AWS account ID

If you are signed into the console, you can view the account ID for your AWS account using the following methods.

To view your AWS account ID

Console

The AWS account ID is displayed when you go to the IAM **Dashboard** in the AWS account section. You can also view your account ID in the navigation bar at the upper right. Choose your user name, and the account ID is displayed above your user name.



AWS CLI

Use the following command to view your user ID, account ID, and your user ARN:

- aws sts get-caller-identity

API

Use the following API to view your user ID, account ID, and your user ARN:

- GetCallerIdentity

Using an alias for your AWS account ID

Your account ID is a 12-digit number that uniquely identifies your account. By default, IAM users in the account sign in using a web URL that includes the account ID. If they don't have the URL, they can provide the account ID on the AWS sign-in page when they sign-in.

Your sign-in page URL has the following format, by default.

```
https://Your_Account_ID.signin.aws.amazon.com/console/
```

Many people find words to be easier to remember than numbers, so creating an alias for your account ID can help your IAM users sign-in easier.

If you create an AWS account alias for your AWS account ID, your sign-in page URL looks like the following example.

```
https://Your_Account_Alias.signin.aws.amazon.com/console/
```

Considerations before creating an account alias

- Your AWS account can have only one alias. If you create a new alias for your AWS account, the new alias overwrites the previous alias, and the URL containing the previous alias stops working.
- The account alias must contain only digits, lowercase letters, and hyphens. For more information on limitations on AWS account entities, see IAM and AWS STS quotas.
- The account alias must be unique across all Amazon Web Services products within a given network *partition*.

  A *partition* is a group of AWS Regions. Each AWS account is scoped to one partition.

  The following are the supported partitions:

  - `aws` - AWS Regions
  - `aws-cn` - China Regions

- `aws-us-gov` - AWS GovCloud (US) Regions

> **Note**
>
> Account aliases are not secrets, and they will appear in your public-facing sign-in page URL. Do not include any sensitive information in your account alias.
> The original URL containing your AWS account ID remains active and can be used after you create your AWS account alias.

For instructions on setting up or removing an alias, see Creating an account alias and Deleting an account alias.

Creating an account alias

To perform the following steps, you must have at least the following IAM permissions:

- `iam:ListAccountAliases`
- `iam:CreateAccountAlias`

To create an AWS account alias

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Dashboard**.
3. In the **AWS Account** section, next to **Account Alias**, choose **Create**. If an alias already exists, then choose **Edit**.
4. In the dialog box, enter the name you want to use for your alias, then choose **Save changes**.

AWS CLI

Run the following command:

- `aws iam create-account-alias`

API

To create an alias for your AWS Management Console sign-in page URL, call the following operation:

- `CreateAccountAlias`

Deleting an account alias

To perform the following steps, you must have at least the following IAM permissions:

- `iam:ListAccountAliases`
- `iam:DeleteAccountAlias`

To delete an account alias

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Dashboard**.
3. In the **AWS Account** section, next to **Account Alias**, choose **Delete**.

AWS CLI

To delete an AWS account ID alias, run the following command:

- `aws iam delete-account-alias`

To confirm that the account alias is deleted, attempt to display your AWS account ID alias, by running the following command:

- `aws iam list-account-aliases`

API

To delete an AWS account ID alias, call the following operation:

- DeleteAccountAlias

To confirm that the account alias is deleted attempt to display your AWS account ID alias, by calling the following operation:

- ListAccountAliases

> **Note**
>
> After deleting your account alias, the only sign-in URL for your account is based off your account ID. Any attempts to connect to the alias URL will fail and are not redirected.

Plan access to your AWS account

When setting up AWS, plan how you intend people to access your AWS account and resources. A well-designed identity management solution keeps your account secure.

Identity sources

According to IAM best practices, we recommend that human users and workloads use temporary credentials to access your AWS resources. You grant temporary credentials through IAM roles. Both users federated into IAM and user in IAM Identity Center (either federated or created in the IAM Identity Center directory) use IAM roles to access resources.

Before you start using AWS, plan how to set up your identities by using one of the following approaches:

- Enabling IAM Identity Center with AWS Organizations and adding users in IAM Identity Center directly to the organizational directory.

  For instructions on adding users directly to the IAM Identity Center organizational directory, see Add users
- Federating your existing external identity provider with either IAM Identity Center or IAM.

  For instructions on federating an external identity provider to the IAM Identity Center organizational directory, see the appropriate Getting started tutorial.

Access management

Identify the AWS resources and services that your users need. Then define the access permissions and policies for each user, group, or role.

- If you use IAM Identity Center, an IAM identity provider, IAM roles, and permissions policies are created in each AWS account in your organization. These roles and permissions match what you specify when you assign people or groups to applications or AWS accounts.

  For more information, see Assign user access and Set up single sign-on access to your applications.
- You can use account access manager — an IAM feature that lets you assign existing IAM roles you create and manage to IAM Identity Center users and groups. Account access manager gives you access to the full IAM role feature set. You can use it alongside permission sets or on its own.
- If you federate your identity provider directly with IAM in your AWS account, you must create a role for your users to assume. The role needs a trust policy and a permissions policy. The trust policy states who can assume the role. The permissions policy states which AWS actions and resources the role is allowed or denied access to.

  For more information, see Identity providers and federation into AWS

For instructions on adding an extra layer of security for sign-in, see Use multi-factor authentication with your identities.

Use cases for IAM users

IAM users that you create in your AWS account have long-term credentials that you manage directly.

When it comes to managing access in AWS, IAM users are generally not the best choice. There are a few key reasons why you should avoid relying on IAM users for most of your use cases.

First, IAM users are designed for individual accounts, so they don't scale well as your organization grows. Managing permissions and security for a large number of IAM users can quickly become a challenge.

IAM users also lack the centralized visibility and auditing capabilities that you get with other AWS identity management solutions. This can make it more challenging to maintain security and regulatory compliance.

Finally, implementing security best practices like multi-factor authentication, password policies, and role separation is much easier with more scalable identity management approaches.

Instead of relying on IAM users, we recommend using more robust solutions like IAM Identity Center with AWS Organizations, or federated identities from external providers. These options will give you better control, security, and operational efficiency as your AWS environment grows.

As a result, we recommend that you only use IAM users for use cases not supported by federated users.

The following list identifies the specific use cases that require long-term credentials with IAM users in AWS. You can use IAM to create these IAM users under the umbrella of your AWS account, and use IAM to manage their permissions.

- Emergency access to your AWS account
- Workloads that can't use IAM roles
  - AWS CodeCommit access
  - Amazon Keyspaces (for Apache Cassandra) access
- Third-party AWS clients
- AWS IAM Identity Center isn't available for your account and you have no other identity provider

Create an IAM user for emergency access

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application.

Having an IAM user for emergency access is one of the recommended reasons to create an IAM user so that you can access your AWS account if your identity provider isn't accessible.

> **Note**
>
> As a security best practice, we recommend that you provide access to your resources through identity federation instead of creating IAM users. For information about specific situations where an IAM user is required, see When to create an IAM user (instead of a role).

To create an IAM user for emergency access

Minimum permissions

To perform the following steps, you must have at least the following IAM permissions:

- `access-analyzer:ValidatePolicy`
- `iam:AddUserToGroup`
- `iam:AttachGroupPolicy`
- `iam:CreateGroup`
- `iam:CreateLoginProfile`
- `iam:CreateUser`
- `iam:GetAccountPasswordPolicy`
- `iam:GetLoginProfile`
- `iam:GetUser`
- `iam:ListAttachedGroupPolicies`
- `iam:ListAttachedUserPolicies`
- `iam:ListGroupPolicies`
- `iam:ListGroups`
- `iam:ListGroupsForUser`
- `iam:ListPolicies`
- `iam:ListUserPolicies`
- `iam:ListUsers`

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, select **Users** and then select **Create user**.

> **Note**
>
> If you have IAM Identity Center enabled, the AWS Management Console displays a reminder that it's best to manage users' access in IAM Identity Center. In this procedure, the IAM user you create is specifically for use only when your identity provider is unavailable.

4. On the **Specify user details** page, under **User details**, in **User name**, enter the name for the new user. This is their sign-in name for AWS. For this example, enter **`EmergencyAccess`**.

> **Note**
>
> User names can be a combination of up to 64 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), underscore (_), and hyphen (-). Names must be unique within an account. They are not distinguished by case. For example, you cannot create two users named TESTUSER and testuser. When a user name is used in a policy or as part of an ARN, the name is case sensitive. When a user name appears to customers in the console, such as during the sign-in process, the user name is case insensitive.

5. Choose the checkbox next to **Provide user access to the AWS Management Console–*optional*** and then choose **I want to create an IAM user**.
6. Under **Console password**, select **Autogenerated password**.
7. Clear the checkbox next to **User must create a new password at next sign-in (recommended)**. Because this IAM user is for emergency access, a trusted administrator retains the password and only provides it when needed.
8. On the **Set permissions** page, under **Permissions options**, select **Add user to group**. Then, under **User groups**, select **Create group**.
9. On the **Create user group** page, in **User group name**, enter **`EmergencyAccessGroup`**. Then, under **Permissions policies**, select **AdministratorAccess**.
10. Choose **Create user group** to return to the **Set permissions** page.
11. Under **User groups**, select the name of the **`EmergencyAccessGroup`** you created previously.
12. Choose **Next** to proceed to the **Review and create** page.

13. On the **Review and create** page, review the list of user group memberships to be added to the new user. When you are ready to proceed, select **Create user**.
14. On the **Retrieve password** page, select **Download .csv file** to save a .csv file with the user credential information (Connection URL, username, and password).
15. Save this file to use if you need to sign-in to IAM and don't have access to your identity provider.

The new IAM user is displayed in the **Users** list. Select the **User name** link to view the user details.

AWS CLI

1. Create a user named **`EmergencyAccess`**.

   - aws iam create-user

   ```
   aws iam create-user \
      --user-name EmergencyAccess
   ```

2. (Optional) Give the user access to the AWS Management Console. This requires a password. To create a password for an IAM user you can use the `--cli-input-json` parameter to pass a JSON file that contains the password.You must also give the user the URL of your account's sign-in page.

   - aws iam create-login-profile

   ```
   aws iam create-login-profile \
      --generate-cli-skeleton > create-login-profile.json
   ```

   - Open the `create-login-profile.json` file in a text editor and enter a password that complies with your password policy, then save the file. For example:

   ```
   {
    "UserName": "EmergencyAccess",
    "Password": "Ex@3dRA0djs",
    "PasswordResetRequired": false
   ```

```
}
```

- Use the `aws iam create-login-profile` command again, passing the `--cli-input-json` parameter to specify your JSON file.

```
aws iam create-login-profile \
   --cli-input-json file://create-login-profile.json
```

> **Note**
>
> If the password you provided in the JSON file violates your account's password policy, you will receive a `PasswordPolicyViolation` error. If this happens, review the password policy for your account and update the password in the JSON file to comply with the requirements.

3. Create the **`EmergencyAccessGroup`**, attach the AWS managed policy `AdministratorAccess` to the group, and add the **`EmergencyAccess`** user to the group.

> **Note**
>
> An *AWS managed policy* is a standalone policy that is created and administered by AWS. Each policy has its own Amazon Resource Name (ARN) that includes the policy name. For example, `arn:aws:iam::aws:policy/IAMReadOnlyAccess` is an AWS managed policy. For more information about ARNs, see IAM ARNs. For a list of AWS managed policies for AWS services, see AWS managed policies.

- aws iam create-group

```
aws iam create-group \
   --group-name EmergencyAccessGroup
```

- aws iam attach-group-policy

```
aws iam attach-group-policy \
   --policy-arn arn:aws:iam::aws:policy/AdministratorAccess \
   --group-name >EmergencyAccessGroup
```

- aws iam add-user-to-group

```
aws iam add-user-to-group \
   --user-name EmergencyAccess \
   --group-name EmergencyAccessGroup
```

- Run the aws iam get-group command to list the**EmergencyAccessGroup** and its members.

```
aws iam get-group \
   --group-name EmergencyAccessGroup
```

Create an IAM user for workloads that can't use IAM roles

> **Important**
>
> As a best practice, we recommend you require your human users to use temporary credentials when accessing AWS.
> Alternatively, you can manage your user identities, including your administrative user, with AWS IAM Identity Center. We recommend you use IAM Identity Center to manage access to your accounts and permissions within those accounts. If you are using an external identity provider, you can also configure the access permissions for user identities in IAM Identity Center.

If your use case requires IAM users with programmatic access and long-term credentials, we recommend that you establish procedures to update access keys when needed. For more information, see Update access keys.

To perform some account and service management tasks, you must sign in using root user credentials. To view the tasks that require you to sign in as the root user, see Tasks that require root user credentials.

To create an IAM user for workloads that can't use IAM roles

Minimum permissions

To perform the following steps, you must have at least the following IAM permissions:

- `iam:AddUserToGroup`
- `iam:AttachGroupPolicy`
- `iam:CreateAccessKey`
- `iam:CreateGroup`
- `iam:CreateServiceSpecificCredential`
- `iam:CreateUser`
- `iam:GetAccessKeyLastUsed`
- `iam:GetAccountPasswordPolicy`
- `iam:GetAccountSummary`
- `iam:GetGroup`
- `iam:GetLoginProfile`
- `iam:GetPolicy`
- `iam:GetRole`
- `iam:GetUser`
- `iam:ListAccessKeys`
- `iam:ListAttachedGroupPolicies`
- `iam:ListAttachedUserPolicies`
- `iam:ListGroupPolicies`
- `iam:ListGroups`
- `iam:ListGroupsForUser`
- `iam:ListInstanceProfilesForRole`
- `iam:ListMFADevices`
- `iam:ListPolicies`

- `iam:ListRoles`
- `iam:ListRoleTags`
- `iam:ListSSHPublicKeys`
- `iam:ListServiceSpecificCredentials`
- `iam:ListSigningCertificates`
- `iam:ListUserPolicies`
- `iam:ListUserTags`
- `iam:ListUsers`
- `iam:UploadSSHPublicKey`
- `iam:UploadSigningCertificate`

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users** and then choose **Create user**.
4. On the **Specify user details** page, do the following:
   a. For **User name**, type ***`WorkloadName`***. Replace ***`WorkloadName`*** with the name of the workload that will be using the account.
   b. Choose **Next**.
5. (Optional) On the **Set Permissions** page, do the following:
   a. Choose **Add user to group**.
   b. Choose **Create group**.
   c. In the **Create user group** dialog box, for **User group name** type a name that represents the use of the workloads in the group. For this example, use the name **`Automation`**.
   d. Under **Permissions policies** select the checkbox for the **PowerUserAccess** managed policy.

> **Tip**
>
> Enter *Power* into the **Permissions policies** search box to quickly find the managed policy.

e. Choose **Create user group**.

f. Back on the page with the list of IAM groups, select the checkbox for your new user group. Choose **Refresh** if you don't see the new user group in the list.

g. Choose **Next**.

6. (Optional) In the **Tags** section, add metadata to the user by attaching tags as key-value pairs. For more information, see Tags for AWS Identity and Access Management resources.

7. Verify the user group memberships for the new user. When you are ready to proceed, choose **Create user**.

8. A status notification appears informing you that the user was created successfully. Select **View user** to go to the user details page

9. Select the **Security credentials** tab. Then create the credentials needed for the workload.

- **Access keys**–Select **Create access key** to generate and download access keys for the user.

> **Important**
>
> This is your only opportunity to view or download the secret access keys, and you must provide this information to your users before they can use the AWS API. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret keys again after this step.**

- **SSH public keys for AWS CodeCommit**–Select **Upload SSH public key** to upload an SSH public key so that the user can communicate with CodeCommit repositories over SSH.
- **HTTPS Git credentials for AWS CodeCommit**–Select **Generate credentials** to generate a unique set of user credentials to use with Git repositories. Select **Download credentials** to save the user name and password to a .csv file. This is the only time that information is available. If you forget or lose the password you will need to reset it.
- **Credentials for Amazon Keyspaces (for Apache Cassandra)**–Select **Generate credentials** to generate a service-specific user credentials to use with Amazon Keyspaces. Select **Download credentials** to save the user name and password to a .csv file. This is the only

time that information is available. If you forget or lose the password you will need to reset it.

> **Important**
>
> Service-specific credentials are long-term credentials associated with a specific IAM user and can only be used for the service they were created for. To give IAM roles or federated identities permissions to access all your AWS resources using temporary credentials, use AWS authentication with the SigV4 authentication plugin for Amazon Keyspaces. For more information see, Using temporary credentials to connect to Amazon Keyspaces (for Apache Cassandra) using an IAM role and the SigV4 plugin in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

- **X.509 Signing certificates**–Select **Create X.509 Certificate** if you need to make secure SOAP-protocol requests and are in a Region that's not supported by AWS Certificate Manager. ACM is the preferred tool to provision, manage, and deploy your server certificates. For more information about using ACM, see the *AWS Certificate Manager User Guide*.

You have created a user with programmatic access and configured it with the **PowerUserAccess** job function. This user's permissions policy grants full access to every service except for IAM and AWS Organizations.

You can use this same process to give additional workloads programmatic access to your AWS account resources, if the workloads are unable to assume IAM roles. This procedure used the **PowerUserAccess** managed policy to assign permissions. To follow the best practice of least privilege, consider using a more restrictive policy or creating a custom policy that restricts access to only resources required by the program. To learn about using policies that restrict user permissions to specific AWS resources, see Access management for AWS resources and Example IAM identity-based policies. To add additional users to the user group after it's created, see Edit users in IAM groups.

AWS CLI

1. Create a user named **`Automation`**.

   - aws iam create-user

```
aws iam create-user \
    --user-name Automation
```

2. Create an IAM user group named **AutomationGroup**, attach the AWS managed policy PowerUserAccess to the group, and then add the **Automation** user to the group.

> **Note**
>
> An *AWS managed policy* is a standalone policy that is created and administered by AWS. Each policy has its own Amazon Resource Name (ARN) that includes the policy name. For example, arn:aws:iam::aws:policy/IAMReadOnlyAccess is an AWS managed policy. For more information about ARNs, see IAM ARNs. For a list of AWS managed policies for AWS services, see AWS managed policies.

- aws iam create-group

```
aws iam create-group \
    --group-name AutomationGroup
```

- aws iam attach-group-policy

```
aws iam attach-group-policy \
    --policy-arn arn:aws:iam::aws:policy/PowerUserAccess \
    --group-name AutomationGroup
```

- aws iam add-user-to-group

```
aws iam add-user-to-group \
    --user-name Automation \
    --group-name AutomationGroup
```

- Run the aws iam get-group command to list the **AutomationGroup** and its members.

```
aws iam get-group \
     --group-name AutomationGroup
```

3. Create the security credentials needed for the workload.

- **Create access keys for testing**–aws iam create-access-key

```
aws iam create-access-key \
    --user-name Automation
```

The output of this command displays the secret access key and the access key ID. Record and store this information in a secure location. If these credentials are lost, they can't be recovered, and you must create a new access key.

> **Important**
>
> These IAM user access keys are long-term credentials that present a security-risk to your account. After you have completed testing, we recommend that you delete these access keys. If you have scenarios in which you are considering access keys, investigate whether you can enable MFA for your workload IAM user and use aws sts get-session-token to obtain temporary credentials for the session instead of using IAM access keys.

- **Upload SSH public keys for AWS CodeCommit**–aws iam upload-ssh-public-key

The following example assumes that you have your SSH public keys stored in the file `sshkey.pub`.

```
aws upload-ssh-public-key \
    --user-name Automation \
    --ssh-public-key-body file://sshkey.pub
```

- **Upload an X.509 signing certificate**–aws iam upload-signing-certificate

Upload an X.509 certificate if you need to make secure SOAP-protocol requests and are in a Region that's not supported by AWS Certificate Manager. ACM is the preferred tool to provision, manage, and deploy your server certificates. For more information about using ACM, see the *AWS Certificate Manager User Guide*.

The following example assumes that you have your X.509 signing certificate stored in the file `certificate.pem`.

```
aws iam upload-signing-certificate \
--user-name Automation \
--certificate-body file://certificate.pem
```

You can use this same process to give additional workloads programmatic access to your AWS account resources, if the workloads are unable to assume IAM roles. This procedure used the **PowerUserAccess** managed policy to assign permissions. To follow the best practice of least privilege, consider using a more restrictive policy or creating a custom policy that restricts access to only resources required by the program. To learn about using policies that restrict user permissions to specific AWS resources, see Access management for AWS resources and Example IAM identity-based policies. To add additional users to the user group after it's created, see Edit users in IAM groups.

Use multi-factor authentication with your identities

Using multi-factor authentication (MFA) with your identities is another IAM best practice. MFA is an additional security layer that requires users to provide additional authentication factors after providing their username and password to verify their identity. It significantly enhances security by making it much harder for attackers to gain unauthorized access, even if a user's password is compromised. MFA is widely adopted as a best practice for securing access to online accounts, cloud services, and other sensitive resources. AWS supports MFA for root user, IAM users, users in IAM Identity Center, Builder ID, and federated users. For additional security, you can create policies that require MFA be configured before allowing a user to access resources or take specific actions and attach these policies to your IAM roles. IAM Identity Center comes preconfigured with MFA turned on by default so that all users in IAM Identity Center must sign in with MFA in addition to their user name and password.

> **Note**
>
> All AWS account types (standalone, management, and members accounts) require MFA to be configured for their root user. Users must register MFA within 14 days of their first sign-in attempt to access the AWS Management Console if MFA is not already enabled.

For more information, see Configure MFA in IAM Identity Center and AWS Multi-factor authentication in IAM.

Prepare for least-privilege permissions

Using *least-privilege permissions* is an IAM best practice recommendation. The concept of least-privilege permissions is to grant users the permissions required to perform a task and no additional permissions. As you get set up, consider how you are going to support least-privilege permissions. The root user, the administrative user, and the emergency access IAM user have powerful permissions that aren't required for everyday tasks. While you are learning about AWS and testing out different services we recommend that you create at least one additional user in IAM Identity Center with lesser permissions that you can use in different scenarios. You can use IAM policies to define the actions that can be taken on specific resources under specific conditions and then connect to those resources with your lesser privileged account.

If you are using IAM Identity Center, consider using IAM Identity Center permissions sets to get started. To learn more, see Create a permission set in the *IAM Identity Center User Guide*.

If you aren't using IAM Identity Center, use IAM roles to define the permissions for different IAM entities. To learn more, see IAM role creation.

Both IAM roles and IAM Identity Center permissions sets can use AWS managed policies based on job functions. For details on the permissions granted by these policies, see AWS managed policies for job functions.

> **Important**
>
> Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for use by all AWS customers. After getting set up, we recommend that you use IAM Access Analyzer to generate least-privilege policies

based on your access activity that's logged in AWS CloudTrail. For more information about policy generation, see IAM Access Analyzer policy generation.

When you are getting started, we recommend that you use AWS managed policies to grant permissions. After a predefined sample period of activity (such as 90 days) has passed, you can review the services that people and workloads have accessed. Then you can create a new customer managed policy with reduced permissions to replace the AWS managed policy. The new policy should include only the services that were accessed during the sample period. Update your permissions to remove the AWS managed policy and attach the new customer managed policy you created.

For instructions on refining permissions, see Reviewing last accessed information for your AWS account. After you identify unused services, see Generating a policy based on access activity.

Reviewing last accessed information for your AWS account

You can view service last accessed information for IAM using the IAM console, AWS CLI, or AWS API. For important information about the data, permissions required, troubleshooting, and supported Regions, see Refine permissions in AWS using last accessed information.

You can view information for the following resource types in IAM. In each case, the information includes allowed services for the given reporting period:

- **IAM user** – View the last time that the user attempted to access each allowed service.
- **IAM group** – View information about the last time that an IAM group member attempted to access each allowed service. This report also includes the total number of members that attempted access.
- **IAM role** – View the last time that someone used the role in an attempt to access each allowed service.
- **Policy** – View information about the last time that a user or role attempted to access each allowed service. This report also includes the total number of entities that attempted access.

> **Note**
>
> Before you view the access information for a resource in IAM, make sure you understand the reporting period, reported entities, and the evaluated policy types for your information. For more details, see the section called "Things to know about last accessed information".

For more information about the last accessed information, see Refine permissions in AWS using last accessed information.

To review last accessed information for an AWS account

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose either **User groups**, **Users**, **Roles**, or **Policies**.
4. Choose any user, user group, role, or policy name to open its **Summary** page and choose the **Last Accessed** tab. View the following information, based on the resource that you chose:

   - **User group** – View the list of services that user group members can access. You can also view when a member last accessed the service, what user group policies they used, and which user group member made the request. Choose the name of the policy to learn whether it is a managed policy or an inline user group policy. Choose the name of the user group member to see all of the members of the user group and when they last accessed the service.
   - **User** – View the list of services that the user can access. You can also view when they last accessed the service, and what policies are currently associated with the user. Choose the name of the policy to learn whether it is a managed policy, an inline user policy, or an inline policy for the user group.
   - **Role** – View the list of services that the role can access, when the role last accessed the service, and what policies were used. Choose the name of the policy to learn whether it is a managed policy or an inline role policy.

- **Policy** – View the list of services with allowed actions in the policy. You can also view when the policy was last used to access the service, and which entity (user or role) used the policy. The **Last accessed** date also includes when access is granted to this policy through another policy. Choose the name of the entity to learn which entities have this policy attached and when they last accessed the service.

5. In the **Service** column of the table, choose the name of one of the services that includes action last accessed information to view a list of management actions that IAM entities have attempted to access. You can view the AWS Region and a timestamp that shows when someone last attempted to perform the action.
6. The **Last accessed** column is displayed for services and management actions of the services that include action last accessed information. Review the following possible results that are returned in this column. These results vary depending on whether a service or action is allowed, was accessed, and whether it is tracked by AWS for last accessed information.

   <number of> days ago

   The number of days since the service or action was used in the tracking period. The tracking period for services is for the last 400 days. The tracking period for Amazon S3 actions started on April 12, 2020. The tracking period for Amazon EC2, IAM, and Lambda actions started on April 7, 2021. The tracking period for all other services began on May 23, 2023. To learn more about the tracking start dates for each AWS Region, see Where AWS tracks last accessed information.

   Not accessed in the tracking period

   The tracked service or action has not been used by an entity in the tracking period.

   It is possible for you to have permissions for an action that doesn't appear in the list. This can happen if the tracking information for the action is not currently included by AWS. You should not make permissions decisions based solely on the absence of tracking information. Instead, we recommend that you use this information to inform and support your overall strategy of granting least privilege. Check your policies to confirm that the level of access is appropriate.

AWS CLI

You can use the AWS CLI to retrieve information about the last time that an IAM resource in your AWS account was used to attempt to access AWS services and Amazon S3, Amazon EC2, IAM, and Lambda actions. An IAM resource can be a user, user group, role, or policy.

- Generate a report for IAM resources in an AWS account. The request must include the ARN of the IAM resource (user, user group, role, or policy) for which you want a report. You can specify the level of granularity that you want to generate in the report to view access details for either services or both services and actions. The request returns a `job-id` that you can then use in the `get-service-last-accessed-details` and `get-service-last-accessed-details-with-entities` operations to monitor the `job-status` until the job is complete.

  - aws iam generate-service-last-accessed-details

  a. Retrieve details about the report using the `job-id` parameter from the previous step.

     - aws iam get-service-last-accessed-details

     This operation returns the following information, based on the type of resource and level of granularity that you requested in the `generate-service-last-accessed-details` operation:

     - **User** – Returns a list of services that the specified user can access. For each service, the operation returns the date and time of the user's last attempt and the ARN of the user.
     - **User group** – Returns a list of services that members of the specified user group can access using the policies attached to the user group. For each service, the operation returns the date and time of the last attempt made by any user group member. It also returns the ARN of that user and the total number of user group members that have attempted to access the service. Use the GetServiceLastAccessedDetailsWithEntities operation to retrieve a list of all of the members.
     - **Role** – Returns a list of services that the specified role can access. For each service, the operation returns the date and time of the role's last attempt and the ARN of the role.

- **Policy** – Returns a list of services for which the specified policy allows access. For each service, the operation returns the date and time that an entity (user or role) last attempted to access the service using the policy. It also returns the ARN of that entity and the total number of entities that attempted access.

b. Learn more about the entities that used user group or policy permissions in an attempt to access a specific service. This operation returns a list of entities with each entity's ARN, ID, name, path, type (user or role), and when they last attempted to access the service. You can also use this operation for users and roles, but it only returns information about that entity.

- aws iam get-service-last-accessed-details-with-entities

c. Learn more about the identity-based policies that an identity (user, user group, or role) used in an attempt to access a specific service. When you specify an identity and service, this operation returns a list of permissions policies that the identity can use to access the specified service. This operation gives the current state of policies and does not depend on the generated report. It also does not return other policy types, such as resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see Policy types or Policy evaluation for requests within a single account.

- aws iam list-policies-granting-service-access

API

You can use the AWS API to retrieve information about the last time that an IAM resource was used to attempt to access AWS services and Amazon S3, Amazon EC2, IAM, and Lambda actions. An IAM resource can be a user, user group, role, or policy. You can specify the level of granularity to generate in the report to view details for either services or both services and actions.

1. Generate a report. The request must include the ARN of the IAM resource (user, user group, role, or policy) for which you want a report. It returns a `JobId` that you can then use in the `GetServiceLastAccessedDetails` and `GetServiceLastAccessedDetailsWithEntities` operations to monitor the `JobStatus` until the job is complete.

   - GenerateServiceLastAccessedDetails

2. Retrieve details about the report using the `JobId` parameter from the previous step.

   - GetServiceLastAccessedDetails

   This operation returns the following information, based on the type of resource and level of granularity that you requested in the `GenerateServiceLastAccessedDetails` operation:

   - **User** – Returns a list of services that the specified user can access. For each service, the operation returns the date and time of the user's last attempt and the ARN of the user.
   - **User group** – Returns a list of services that members of the specified user group can access using the policies attached to the user group. For each service, the operation returns the date and time of the last attempt made by any user group member. It also returns the ARN of that user and the total number of user group members that have attempted to access the service. Use the GetServiceLastAccessedDetailsWithEntities operation to retrieve a list of all of the members.
   - **Role** – Returns a list of services that the specified role can access. For each service, the operation returns the date and time of the role's last attempt and the ARN of the role.
   - **Policy** – Returns a list of services for which the specified policy allows access. For each service, the operation returns the date and time that an entity (user or role) last attempted to access the service using the policy. It also returns the ARN of that entity and the total number of entities that attempted access.

3. Learn more about the entities that used user group or policy permissions in an attempt to access a specific service. This operation returns a list of entities with each entity's ARN, ID, name, path, type (user or role), and when they last attempted to access the service. You can also use this operation for users and roles, but it only returns information about that entity.

   - GetServiceLastAccessedDetailsWithEntities

4. Learn more about the identity-based policies that an identity (user, user group, or role) used in an attempt to access a specific service. When you specify an identity and service, this operation returns a list of permissions policies that the identity can use to access the specified service. This operation gives the current state of policies and does not depend on the generated report. It also does not return other policy types, such as resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see Policy types or Policy evaluation for requests within a single account.

- ListPoliciesGrantingServiceAccess

Generating a policy based on access activity

You can use the access activity recorded in AWS CloudTrail for an IAM user or IAM role to have IAM Access Analyzer generate a customer managed policy to allow access to only the services that specific users and roles need.

When IAM Access Analyzer generates an IAM policy, information is returned to help you to further customize the policy. Two categories of information can be returned when a policy is generated:

- **Policy with action-level information –** For some AWS services, such as Amazon EC2, IAM Access Analyzer can identify the actions found in your CloudTrail events and lists the actions used in the policy it generates. For a list of supported services, see IAM Access Analyzer policy generation services. For some services, IAM Access Analyzer prompts you to add actions for the services to the generated policy.
- **Policy with service-level information –** IAM Access Analyzer uses last accessed information to create a policy template with all of the recently used services. When using the AWS Management Console, we prompt you to review the services and add actions to complete the policy.

To generate a policy based on access activity

In the following procedure we are going to reduce the permissions given to a role to match the usage of a user. When you choose a user, choose a user whose usage exemplifies the role. Many customers set up test user accounts with **PowerUser** permissions and then have them do a specific set of tasks for a short time period to determine what access is necessary to perform those tasks,

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users** and then choose the user name to go to the user details page.

4. On **Permissions** tab, under Generate policy based on CloudTrail events, choose **Generate policy**.
5. On the **Generate policy** page, configure the following items:
   - For **Select time period**, choose **Last 7 days**.
   - For **CloudTrail trail to be analyzed**, select the Region and trail where this user's activity is recorded.
   - Choose **Create and use a new service role**.
6. Choose **Generate policy** then wait until the role is created. Don't refresh or navigate away from the console page until the **Policy generation in progress** notification message appears.
7. After the policy is generated, you must review and customize it as needed with the account IDs and ARNs for resources. In addition, the automatically generated policy might not include the action-level information needed to complete the policy. For more information, see IAM Access Analyzer policy generation.

   For example, you might edit the first statement that includes the `Allow` effect and the `NotAction` element to allow only Amazon EC2 and Amazon S3 actions. To do this, replace it with the statement with the `FullAccessToSomeServices` ID. Your new policy could look like the following example policy.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Sid": "FullAccessToSomeServices",
          "Effect": "Allow",
          "Action": [
              "ec2:*",
              "s3:*"
          ],
          "Resource": "*"
      },
      {
          "Effect": "Allow",
          "Action": [
```

```
                "iam:CreateServiceLinkedRole",
                "iam:DeleteServiceLinkedRole",
                "iam:ListRoles",
                "organizations:DescribeOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

8. To support the best practice of granting least privilege, review and correct any errors, warnings, or suggestions returned during policy validation.
9. To further reduce your policies' permissions to specific actions and resources, view your events in CloudTrail **Event history**. There you can view detailed information about the specific actions and resources that your user has accessed. For more information, see Viewing CloudTrail Events in the CloudTrail Console in the *AWS CloudTrail User Guide*.
10. After reviewing and validating your policy, save it with a descriptive name.
11. Navigate to the **Roles** page and choose the role that people will assume when they perform the tasks permitted by your new policy.
12. Select the **Permissions** tab, and then choose **Add permissions** and select **Attach policies**.
13. On the **Attach permission policies** page, in the **Other permissions policies** list, select the policy you created, then choose **Attach policies**.
14. You are returned to the **Role** details page. There are two policies attached to the role, your previous AWS managed policy, such as **PowerUserAccess**, and your new policy. Select the checkbox for the AWS managed policy and then choose **Remove**. When asked to confirm removal, choose **Remove**.

IAM users, SAML and OIDC federated principals, and workloads who assume this role now have reduced access according to the new policy you created.

AWS CLI

You can use the following commands to generate a policy using the AWS CLI.

To generate a policy

- aws accessanalyzer start-policy-generation

To view a generated policy

- aws accessanalyzer get-generated-policy

To cancel a policy generation request

- aws accessanalyzer cancel-policy-generation

To view a list of policy generation requests

- aws accessanalyzer list-policy-generations

API

You can use the following operations to generate a policy using the AWS API.

To generate a policy

- StartPolicyGeneration

To view a generated policy

- GetGeneratedPolicy

To cancel a policy generation request

- CancelPolicyGeneration

To view a list of policy generation requests

- ListPolicyGenerations

Using search to find IAM resources

As you work through your access findings, you can use the IAM console search page as a faster option for finding IAM resources. You can search for resources using partial resource names or ARNs.

Console

The IAM console search feature can locate any of the following:

- IAM entity names that match your search keywords (for users, groups, roles, identity providers, and policies)
- Tasks that match your search keywords

The IAM console search feature does not return information about IAM Access Analyzer.

Every line in the search result is an active link. For example, you can choose the user name in the search result, which takes you to that user's detail page. Or you can choose an action link, for example **Create user**, to go to the **Create User** page.

> **Note**
>
> Access key search requires you to type the full access key ID in the search box. The search result shows the user associated with that key. From there you can navigate directly to that user's page, where you can manage the access key.

Use the **Search** page in the IAM console to find items related to that account.

To search for items in the IAM console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Search**.
4. In the **Search** box, type your search keywords.
5. Choose a link in the search results list to navigate to the corresponding part of the console.

The following icons identify the types of items that are found by a search:

Icon	Description
	IAM users
	IAM groups
	IAM roles
	IAM policies
	Tasks such as "create user" or "attach policy"
	Results from the keyword **`delete`**

Sample search phrases

You can use the following phrases in the IAM search. Replace terms in italics with the names of the actual IAM users, groups, roles, access keys, policies, or identity providers that you want to locate.

- ***`user_name`*** or ***`group_name`*** or ***`role_name`*** or ***`policy_name`*** or ***`identity_provider_name`***
- ***`access_key`***
- **`add user`** ***`user_name`*** **`to groups`** or **`add users to group`** ***`group_name`***
- **`remove user`** ***`user_name`*** **`from groups`**
- **`delete`** ***`user_name`*** or **`delete`** ***`group_name`*** or **`delete`** ***`role_name`***, or **`delete`** ***`policy_name`***, or **`delete`** ***`identity_provider_name`***
- **`manage access keys`** ***`user_name`***
- **`manage signing certificates`** ***`user_name`***
- **`users`**
- **`manage MFA for`** ***`user_name`***

- **manage password for *user_name***
- **create role**
- **password policy**
- **edit trust policy for role *role_name***
- **show policy document for role *role_name***
- **attach policy to *role_name***
- **create managed policy**
- **create user**
- **create group**
- **attach policy to *group_name***
- **attach entities to *policy_name***
- **detach entities from *policy_name***

Security best practices and use cases in AWS Identity and Access Management

AWS Identity and Access Management (IAM) offers several security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution.

These best practices may not be appropriate or sufficient for your specific environment. Therefore, treat them as helpful considerations rather than requirements.

To get the most out of IAM, take the time to learn the recommended best practices. One effective way to do this is to see how IAM is used in real-world scenarios to work with other AWS services.

Topics

- Security best practices in IAM
- Root user best practices for your AWS account
- Business use cases for IAM

Security best practices in IAM

Follow us on Twitter

To help secure your AWS resources, follow these best practices for AWS Identity and Access Management (IAM).

After you implement these best practices, see Security in IAM and AWS STS for additional security topics including compliance validation, resilience, and infrastructure security.

Topics

- Require human users to use federation with an identity provider to access AWS using temporary credentials
- Require workloads to use temporary credentials with IAM roles to access AWS
- Require multi-factor authentication (MFA)
- Update access keys when needed for use cases that require long-term credentials
- Follow best practices to protect your root user credentials

- Apply least-privilege permissions
- Get started with AWS managed policies and move toward least-privilege permissions
- Use IAM Access Analyzer to generate least-privilege policies based on access activity
- Regularly review and remove unused users, roles, permissions, policies, and credentials
- Use conditions in IAM policies to further restrict access
- Verify public and cross-account access to resources with IAM Access Analyzer
- Use IAM Access Analyzer to validate your IAM policies for secure and functional permissions
- Establish permissions guardrails across multiple accounts
- Use permissions boundaries to delegate permissions management within an account

Require human users to use federation with an identity provider to access AWS using temporary credentials

Human users, also known as *human identities,* are the people, administrators, developers, operators, and consumers of your applications. They must have an identity to access your AWS environments and applications. Human users that are members of your organization are also known as *workforce identities.* Human users can also be external users with whom you collaborate, and who interact with your AWS resources. They can do this through a web browser, client application, mobile app, or interactive command-line tools.

Require your human users to use temporary credentials when accessing AWS. You can use an identity provider for your human users to provide federated access to AWS accounts by assuming roles, which provide temporary credentials. For centralized access management, we recommend that you use AWS IAM Identity Center (IAM Identity Center) to manage access to your accounts and permissions within those accounts. You can manage your user identities with IAM Identity Center, or manage access permissions for user identities in IAM Identity Center from an external identity provider. For more information, see What is AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

You can use account access manager — an IAM feature that lets you assign existing IAM roles across your organization's accounts to IAM Identity Center users and groups. Account access manager gives you access to the full IAM role feature set, including custom trust policies, role tags for ABAC, and configurable role paths. You can use it alongside permission sets or on its own.

For more information about roles, see Roles terms and concepts.

Require workloads to use temporary credentials with IAM roles to access AWS

A *workload* is a collection of resources and code that delivers business value, such as an application or backend process. Your workload can have applications, operational tools, and components that require credentials to make requests to AWS services, such as requests to read data from Amazon S3.

When you're building on an AWS compute service, such as Amazon EC2 or Lambda, AWS delivers the temporary credentials of an IAM role to that compute resource. Applications written using an AWS SDK will discover and use these temporary credentials to access AWS resources, and there is no need to distribute long lived credentials for an IAM user to your workloads running on AWS.

Workloads that run outside of AWS, such as your on-premises servers, servers from other cloud providers, or managed continuous integration and continuous delivery (CI/CD) platforms, can still use temporary credentials. However, you'll need to deliver these temporary credentials to your workload. The following are ways you can deliver temporary credentials to your workloads:

- You can use IAM Roles Anywhere to request temporary AWS credentials for your workload using an X.509 Certificate from your public key infrastructure (PKI).
- You can call the AWS STS `AssumeRoleWithSAML` API to request temporary AWS credentials for your workload using a SAML assertion from an external identity provider (IdP) that is configured within your AWS account.
- You can call the AWS STS `AssumeRoleWithWebIdentity` API to request temporary AWS credentials for your workload using a JSON web token (JWT) from an IdP that is configured within your AWS account.
- You can request temporary AWS credentials from your IoT device using Mutual Transport Layer Security (MTLS) authentication using AWS IoT Core.

Some AWS services also support integrations to deliver temporary credentials to your workloads running outside of AWS:

- Amazon Elastic Container Service (Amazon ECS) Anywhere lets you run Amazon ECS tasks on your own compute resources, and delivers temporary AWS credentials to your Amazon ECS tasks running on those compute resources.

- Amazon Elastic Kubernetes Service Hybrid Nodes lets you join your compute resources running outside of AWS as nodes to an Amazon EKS cluster. Amazon EKS can deliver temporary credentials to the Amazon EKS pods running on your compute resources.
- AWS Systems ManagerHybrid Activations lets you manage your compute resources running outside of AWS using SSM, and delivers temporary AWS credentials to the SSM agent running on your compute resources.

Require multi-factor authentication (MFA)

We recommend using IAM roles for human users and workloads that access your AWS resources so that they use temporary credentials. However, for scenarios in which you need an IAM user or root user in your account, require MFA for additional security. With MFA, users have a device that generates a response to an authentication challenge. Each user's credentials and device-generated response are required to complete the sign-in process. For more information, see AWS Multi-factor authentication in IAM. We recommend that you use phishing-resistant MFA such as passkeys and security keys wherever possible. For more information, see Assign a passkey or security key in the AWS Management Console.

If you use IAM Identity Center for centralized access management for human users, you can use the IAM Identity Center MFA capabilities when your identity source is configured with the IAM Identity Center identity store, AWS Managed Microsoft AD, or AD Connector. For more information about MFA in IAM Identity Center see Multi-factor authentication in the *AWS IAM Identity Center User Guide*.

Update access keys when needed for use cases that require long-term credentials

Where possible, we recommend relying on temporary credentials instead of creating long-term credentials such as access keys. However, for scenarios in which you need IAM users with programmatic access and long-term credentials, we recommend that you update the access keys when needed, such as when an employee leaves your company. We recommend that you use *IAM access last used information* to update and remove access keys safely. For more information, see Update access keys.

There are specific use cases that require long-term credentials with IAM users in AWS. Some of the use cases include the following:

- **Workloads that cannot use IAM roles** – You might run a workload from a location that needs to access AWS. In some situations, you can't use IAM roles to provide temporary credentials, such as for WordPress plugins. In these situations, use IAM user long-term access keys for that workload to authenticate to AWS.
- **Third-party AWS clients** – If you are using tools that don't support access with IAM Identity Center, such as third-party AWS clients or vendors that are not hosted on AWS, use IAM user long-term access keys.
- **AWS CodeCommit access** – If you are using CodeCommit to store your code, you can use an IAM user with either SSH keys or service-specific credentials for CodeCommit to authenticate to your repositories. We recommend that you do this in addition to using a user in IAM Identity Center for normal authentication. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. To give users access to your CodeCommit repositories without configuring IAM users, you can configure the **git-remote-codecommit** utility. For more information about IAM and CodeCommit, see IAM credentials for CodeCommit: Git credentials, SSH keys, and AWS access keys. For more information about configuring the **git-remote-codecommit** utility, see Connecting to AWS CodeCommit repositories with rotating credentials in the *AWS CodeCommit User Guide*.
- **Amazon Keyspaces (for Apache Cassandra) access** – In a situation where you are unable to use users in IAM Identity Center, such as for testing purposes for Cassandra compatibility, you can use an IAM user with service-specific credentials to authenticate with Amazon Keyspaces. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. You can also connect to Amazon Keyspaces using temporary credentials. For more information, see Using temporary credentials to connect to Amazon Keyspaces using an IAM role and the SigV4 plugin in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

Follow best practices to protect your root user credentials

When you create an AWS account, you establish root user credentials to sign in to the AWS Management Console. Safeguard your root user credentials the same way you would protect other sensitive personal information. To better understand how to secure and scale your root user processes, see Root user best practices for your AWS account.

Apply least-privilege permissions

When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. You might start with broad permissions while you explore the permissions that are required for your workload or use case. As your use case matures, you can work to reduce the permissions that you grant to work toward least privilege. For more information about using IAM to apply permissions, see Policies and permissions in AWS Identity and Access Management.

Get started with AWS managed policies and move toward least-privilege permissions

To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they are available for use by all AWS customers. As a result, we recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases. For more information, see AWS managed policies. For more information about AWS managed policies that are designed for specific job functions, see AWS managed policies for job functions.

Use IAM Access Analyzer to generate least-privilege policies based on access activity

To grant only the permissions required to perform a task, you can generate policies based on your access activity that is logged in AWS CloudTrail. IAM Access Analyzer analyzes the services and actions that your IAM roles use, and then generates a fine-grained policy that you can use. After you test each generated policy, you can deploy the policy to your production environment. This ensures that you grant only the required permissions to your workloads. For more information about policy generation, see IAM Access Analyzer policy generation.

Regularly review and remove unused users, roles, permissions, policies, and credentials

You might have IAM users, roles, permissions, policies, or credentials that you no longer need in your AWS account. IAM provides *last accessed information* to help you identify the users, roles,

permissions, policies, and credentials that you no longer need so that you can remove them. This helps you reduce the number of users, roles, permissions, policies, and credentials that you have to monitor. You can also use this information to refine your IAM policies to better adhere to least-privilege permissions. For more information, see Refine permissions in AWS using last accessed information.

Use conditions in IAM policies to further restrict access

You can specify conditions under which a policy statement is in effect. That way, you can grant access to actions and resources, but only if the access request meets specific conditions. For example, you can write a policy condition to specify that all requests must be sent using TLS. You can also use conditions to grant access to service actions, but only if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition.

Verify public and cross-account access to resources with IAM Access Analyzer

Before you grant permissions for public or cross-account access in AWS, we recommend that you verify if such access is required. You can use IAM Access Analyzer to help you preview and analyze public and cross-account access for supported resource types. You do this by reviewing the findings that IAM Access Analyzer generates. These findings help you verify that your resource access controls grant the access that you expect. Additionally, as you update public and cross-account permissions, you can verify the effect of your changes before deploying new access controls to your resources. IAM Access Analyzer also monitors supported resource types continuously and generates a finding for resources that allow public or cross-account access. For more information, see Previewing access with IAM Access Analyzer APIs.

Use IAM Access Analyzer to validate your IAM policies for secure and functional permissions

Validate the policies you create to make sure that they adhere to the IAM policy language (JSON) and IAM best practices. You can validate your policies by using IAM Access Analyzer policy validation. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. As you author new policies or edit existing policies in the console, IAM Access Analyzer provides recommendations to help you refine and validate your policies before you save them. Additionally, we recommend that you review and validate all of your existing policies. For more information, see IAM Access Analyzer

policy validation. For more information about policy checks provided by IAM Access Analyzer, see IAM Access Analyzer policy check reference.

Establish permissions guardrails across multiple accounts

As you scale your workloads, separate them by using multiple accounts that are managed with AWS Organizations. We recommend that you use AWS Organizations service control policies (SCPs) to establish permissions guardrails to control access for all principals (IAM roles and users) across your accounts. We recommend that you use AWS Organizations resource control policies (RCPs) to establish permissions guardrails to control access for AWS resources across your organization. SCPs and RCPs are types of organization policies that you can use to manage permissions in your organization at the AWS organization, organizational unit (OU), or account level.

However, SCPs and RCPs alone are insufficient to grant permissions to principals and resources in your organization. No permissions are granted by SCPs and RCPs. To grant permissions, you must attach identity-based or resource-based policies to IAM users, IAM roles, or the resources in your accounts. For more information, see SRA building blocks — AWS Organizations, accounts, and guardrails.

Use permissions boundaries to delegate permissions management within an account

In some scenarios, you might want to delegate permissions management within an account to others. For example, you could allow developers to create and manage roles for their workloads. When you delegate permissions to others, use *permissions boundaries* to set the maximum permissions that you delegate. A permissions boundary is an advanced feature for using a managed policy to set the maximum permissions that an identity-based policy can grant to an IAM role. A permissions boundary does not grant permissions on its own. For more information, see Permissions boundaries for IAM entities.

Root user best practices for your AWS account

When you first create an AWS account, you begin with a default set of credentials with complete access to all AWS resources in your account. This identity is called the AWS account root user. We strongly recommend you don't access the AWS account root user unless you have a task that requires root user credentials. You need to secure your root user credentials and your account recovery mechanisms to help make sure you don't expose your highly privileged credentials for unauthorized use.

For multiple AWS accounts managed through AWS Organizations, we recommend removing root user credentials from member accounts to help prevent unauthorized use. You can remove the root user password, access keys, signing certificates, and deactivate and delete multi-factor authentication (MFA). Member accounts can't sign in to their root user or perform password recovery for their root user. For more information, see Centrally manage root access for member accounts.

Instead of accessing the root user, create an administrative user for everyday tasks.

- If you have a new AWS account, see Setting up your AWS account.
- For multiple AWS accounts managed through AWS Organizations, see Set up AWS account access for an IAM Identity Center administrative user.

With your administrative user, you can then create additional identities for users that need access to resources in your AWS account. We strongly recommend you require users to authenticate with temporary credentials when accessing AWS.

- For a single, standalone AWS account, use IAM roles to create identities in your account with specific permissions. Roles are intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials, such as a password or access keys, associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session. Unlike IAM roles, IAM users have long-term credentials such as passwords and access keys. Where possible, best practices recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys.
- For multiple AWS accounts managed through AWS Organizations, use IAM Identity Center workforce users. With IAM Identity Center, you can centrally manage users across your AWS accounts and permissions within those accounts. Manage your user identities with IAM Identity Center or from an external identity provider. For more information, see What is AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

Topics

- Secure your root user credentials to prevent unauthorized use
- Use a strong root user password to help protect access
- Secure your root user sign-in with multi-factor authentication (MFA)
- Don't create access keys for the root user
- Use multi-person approval for root user sign-in wherever possible

- Use a group email address for root user credentials
- Restrict access to account recovery mechanisms
- Secure your AWS Organizations account root user credentials
- Monitor access and usage

Secure your root user credentials to prevent unauthorized use

Secure your root user credentials and use them for only the tasks that require them. To help prevent unauthorized use, don't share your root user password, MFA, access keys, CloudFront key pairs, or signing certificates with anyone, except those that have a strict business need to access the root user.

Don't store the root user password with tools that depend on AWS services in an account that is accessed using that same password. If you lose or forget your root user password, you will not be able to access these tools. We recommend that you prioritize resiliency and consider requiring two or more people to authorize access to the storage location. Access to the password or its storage location should be logged and monitored.

Use a strong root user password to help protect access

We recommend that you use a password that is strong and unique. Tools such as password managers with strong password generation algorithms can help you achieve these goals. AWS requires that your password meet the following conditions:

- It must have a minimum of 8 characters and a maximum of 128 characters.
- It must include a minimum of three of the following mix of character types: uppercase, lowercase, numbers, and ! @ # $ % ^ & * () <> [] {} | _+-= symbols.
- It must not be identical to your AWS account name or email address.

For more information, see Change the password for the AWS account root user.

Secure your root user sign-in with multi-factor authentication (MFA)

Because a root user can perform privileged actions, it's crucial to add MFA for the root user as a second authentication factor in addition to the email address and password as sign-in credentials. You can register up to eight MFA devices of any combination of the currently supported MFA types with your AWS account root user.

We strongly recommend enabling multiple MFA devices for your root user credentials to provide additional flexibility and resiliency in your security strategy. All AWS account types (standalone, management, and member accounts) require MFA to be configured for their root user. Users must register MFA within 14 days of their first sign-in attempt to access the AWS Management Console if MFA is not already enabled.

- FIDO Certified hardware security keys are provided by third-party providers. For more information, see Enable a FIDO security key for the AWS account root user.
- A hardware device that generates a six-digit numeric code based on the time-based one-time password (TOTP) algorithm. For more information, see Enable a hardware TOTP token for the AWS account root user.
- A virtual authenticator application that runs on a phone or other device and emulates a physical device. For more information, see Enable a virtual MFA device for your AWS account root user.

Don't create access keys for the root user

Access keys let you run commands in the AWS Command Line Interface (AWS CLI) or use API operations from one of the AWS SDKs. We strongly recommend that you do not create access keys for your root user because the root user has full access to all AWS services and resources in the account, including billing information.

Since only a few tasks require the root user and you typically perform those tasks infrequently, we recommend signing in to the AWS Management Console to perform root user tasks. For programmatic access, instead of creating access keys for the root user, use the `aws login` command with your root credentials to authenticate to the AWS CLI and SDKs. This approach provides temporary, automatically rotated credentials that enhance your security posture while eliminating the need to manage long-term access keys. Before creating access keys, review the Alternatives to long-term access keys.

Use multi-person approval for root user sign-in wherever possible

Consider using multi-person approval to make sure that no one person can access both MFA and password for the root user. Some companies add an additional layer of security by setting up one group of administrators with access to the password, and another group of administrators with access to MFA. One member from each group must come together to sign in as the root user.

Use a group email address for root user credentials

Use an email address that is managed by your business and forwards received messages directly to a group of users. If AWS must contact the owner of the account, this approach reduces the risk of delays in responding, even if individuals are on vacation, out sick, or have left the business. The email address used for the root user should not be used for other purposes.

Restrict access to account recovery mechanisms

Make sure you develop a process to manage root user credential recovery mechanisms in case you need access to it during an emergency such as takeover of your administrative account.

- Make sure you have access to your root user email inbox so that you can reset a lost or forgotten root user password.
- If MFA for your AWS account root user is lost, damaged, or not working, you can sign in using another MFA registered to the same root user credentials. If you lost access to all your MFAs, you need both the phone number and the email used to register your account, to be up to date and accessible to recover your MFA. For details, see Recovering a root user MFA device.
- If you choose not to store your root user password and MFA, then the phone number registered in your account can be used as an alternate way to recover root user credentials. Make sure you have access to the contact phone number, keep the phone number updated, and limit who has access to manage the phone number.

No one person should have access to both the email inbox and phone number since both are verification channels to recover your root user password. It is important to have two groups of individuals managing these channels. One group having access to your primary email address and another group having access to the primary phone number to recover access to your account as root user.

Secure your AWS Organizations account root user credentials

As you move to a multi-account strategy with AWS Organizations, each of your AWS accounts has its own root user credentials that you need to secure. The account you use to create your organization is the **management account** and the rest of the accounts in your organization are **member accounts**.

Secure root user credentials for the management account

AWS requires that you register MFA for the root user of your organization's management account. MFA registration must be completed during the first sign-in attempt or within the 14-day grace period. If MFA is not enabled within this time, registration will be required before you can access the AWS Management Console. For more information, see Multi-factor authentication for AWS account root user.

Secure root user credentials for member accounts

If you use AWS Organizations to manage multiple accounts, there are two strategies that you can take to secure root user access in your AWS Organizations.

- Centralize root access and remove root user credentials from member accounts. Remove the root user credentials, access keys, signing certificates, and deactivate and delete multi-factor authentication (MFA). When this strategy is used, member accounts can't sign in to their root user or perform password recovery for their root user. For more information, see Centrally manage root access for member accounts.
- Secure root user credentials of your AWS Organizations accounts with MFA to enhance account security. For more information, see Multi-factor authentication for AWS account root user.

For details, see Accessing member accounts in your organization in the *AWS Organizations User Guide*.

Set preventative security controls in AWS Organizations using a service control policy (SCP)

If the member accounts in your organization have root user credentials enabled, you can apply an SCP to restrict access to member account root user. Denying all root user actions in your member accounts, except for certain root-only actions, helps prevent unauthorized access. For details, see Use an SCP to restrict what the root user in your member accounts can do.

Monitor access and usage

We recommend you use your current tracking mechanisms to monitor, alert, and report the sign in and use of root user credentials, including alerts that announce root user sign-in and usage. The following services can help to make sure that root user credential usage is tracked and perform security checks that can help prevent unauthorized use.

Note

CloudTrail logs different sign-in events for the root user and privileged root user sessions. These privileged sessions allow tasks that require root user credentials to be performed in member accounts in your organization. You can use the sign-in event to identify the actions taken by the management account or a delegated administrator using `sts:AssumeRoot`. For more information, see Track privileged tasks in CloudTrail.

- If you want to be notified about root user sign-in activity in your account, you can leverage Amazon CloudWatch to create an Events rule that detects when root user credentials are used and triggers a notification to your security administrator. For details, see Monitor and notify on AWS account root user activity.
- If you want to set up notifications to alert you of approved root user actions, you can leverage Amazon EventBridge along with Amazon SNS to write an EventBridge rule to track root user usage for the specific action and notify you using an Amazon SNS topic. For an example, see Send a notification when an Amazon S3 object is created.
- If you already using GuardDuty as your threat detection service, you can extend its capability to notify you when root user credentials are being used in your account.

Alerts should include, but not be limited to, the email address for the root user. Have procedures in place for how to respond to alerts so that personnel who receive a root user access alert understand how to validate that root user access is expected, and how to escalate if they believe that a security incident is in progress. For an example of how to configure alerts, see Monitor and notify on AWS account root user activity.

Evaluate root user MFA compliance

The following services can help evaluate MFA compliance for root user credentials.

MFA-related rules return noncompliant if you follow the best practice of removing root user credentials.

We recommend removing root user credentials from member accounts in your organization to help prevent unauthorized use. After you remove root user credentials, including MFA, these member accounts are evaluated as **not applicable**.

- AWS Config provides rules to monitor compliance with root user best practices. You can use AWS Config managed rules to help you enforce MFA for root user credentials. AWS Config can also identify access keys for the root user.
- Security Hub CSPM provides you with a comprehensive view of your security state in AWS and helps you assess your AWS environment against security industry standards and best practices, such as having MFA on the root user and not having root user access keys. For details on the rules available, see AWS Identity and Access Management controls in the *Security Hub CSPM User Guide*.
- Trusted Advisor provides a security check so you know if MFA isn't enabled on the root user account. For more information, see MFA on Root Account in the *AWS Support User Guide*.

If you need to report a security issue on your account, see Report Suspicious Emails or Vulnerability Reporting. Alternatively, you can Contact AWS for assistance and additional guidance.

Business use cases for IAM

A simple business use case for IAM can help you understand basic ways you might implement the service to control the AWS access that your users have. The use case is described in general terms, without the mechanics of how you'd use the IAM API to achieve the results you want.

This use case looks at two typical ways a fictional company called Example Corp might use IAM. The first scenario considers Amazon Elastic Compute Cloud (Amazon EC2). The second considers Amazon Simple Storage Service (Amazon S3).

For more information about using IAM with other services from AWS, see AWS services that work with IAM.

Topics

- Initial setup of example corp
- Use case for IAM with Amazon EC2
- Use case for IAM with Amazon S3

Initial setup of example corp

Nikki Wolf and Mateo Jackson are the founders of Example Corp. Upon starting the company, they create an AWS account and set up AWS IAM Identity Center (IAM Identity Center) to create

administrative accounts to use with their AWS resources. When you set up account access for the administrative user, IAM Identity Center creates a corresponding IAM role. This role, which is controlled by IAM Identity Center, is created in the relevant AWS account, and the policies specified in the **AdministratorAccess** permission set are attached to the role.

Since they now have administrator accounts, Nikki and Mateo no longer need to use their root user to access their AWS account. They plan to only use the root user to complete the tasks that only the root user can perform. After reviewing the security best practices they configure multi-factor authentication (MFA) for their root user credentials and decide how to safeguard their root user credentials.

As their company grows, they hire employees to work as developers, admins, testers, managers, and system administrators. Nikki is in charge of operations, while Mateo manages the engineering teams. They set up an Active Directory Domain Server to manage the employees accounts and manage access to internal company resources.

To give their employees access to AWS resources, they use IAM Identity Center to connect their company's Active Directory to their AWS account.

Because they connected Active Directory to IAM Identity Center, the users, group, and group membership are synchronized and defined. They must assign permission sets and roles to the different groups to give the users the correct level of access to AWS resources. They use AWS managed policies for job functions in the AWS Management Console to create these permissions sets:

- *Administrator*
- *Billing*
- *Developers*
- *Network administrators*
- *Database administrators*
- *System administrators*
- *Support users*

Then they assign these permissions sets to the roles assigned to their Active Directory groups.

For a step-by-step guide describing the initial configuration of IAM Identity Center, see Getting started in the *AWS IAM Identity Center User Guide*. For more information about provisioning IAM

Identity Center user access, see Single sign-on access to AWS accounts in the *AWS IAM Identity Center User Guide*.

Use case for IAM with Amazon EC2

A company like Example Corp typically uses IAM to interact with services like Amazon EC2. To understand this part of the use case, you need a basic understanding of Amazon EC2. For more information about Amazon EC2, see the Amazon EC2 User Guide.

Amazon EC2 permissions for the user groups

To provide "perimeter" control, Nikki attaches a policy to the AllUsers user group. This policy denies any AWS request from a user if the originating IP address is outside Example Corp's corporate network.

At Example Corp, different IAM groups require different permissions:

- **System administrators** – Need permission to create and manage AMIs, instances, snapshots, volumes, security groups, and so on. Nikki attaches the `AmazonEC2FullAccess` AWS managed policy to the SysAdmins user group that gives members of the group permission to use all the Amazon EC2 actions.
- **Developers** – Need the ability to work with instances only. Mateo therefore creates and attaches a policy to the Developers user group that allows developers to call `DescribeInstances`, `RunInstances`, `StopInstances`, `StartInstances`, and `TerminateInstances`.

  > **Note**
  >
  > Amazon EC2 uses SSH keys, Windows passwords, and security groups to control who has access to the operating system of specific Amazon EC2 instances. There's no method in the IAM system to allow or deny access to the operating system of a specific instance.

- **Support users** – Should not be able to perform any Amazon EC2 actions except listing the Amazon EC2 resources currently available. Therefore, Nikki creates and attaches a policy to the Support users group that only lets them call Amazon EC2 "Describe" API operations.

For examples of what these policies might look like, see Example IAM identity-based policies and Using AWS Identity and Access Management in the *Amazon EC2 User Guide*.

User's job function change

At some point, one of the developers, Paulo Santos, changes job functions and becomes a manager. As a manager, Paulo becomes part of the Support users group so that he can open support cases for his developers. Mateo moves Paulo from the Developers user group to the Support users group. As a result of this move, his ability to interact with Amazon EC2 instances is limited. He can't launch or start instances. He also can't stop or terminate existing instances, even if he was the user who launched or started the instance. He can list only the instances that Example Corp users have launched.

Use case for IAM with Amazon S3

Companies like Example Corp would also typically use IAM with Amazon S3. John has created an Amazon S3 bucket for the company called *amzn-s3-demo-bucket*.

Creation of other users and user groups

As employees, Zhang Wei and Mary Major each need to be able to create their own data in the company's bucket. They also need to read and write shared data that all developers work on. To enable this, Mateo logically arranges the data in amzn-s3-demo-bucket using an Amazon S3 key prefix scheme as shown in the following figure.

```
/amzn-s3-demo-bucket
    /home
        /zhang
        /major
    /share
        /developers
        /managers
```

Mateo divides the `/amzn-s3-demo-bucket` into a set of home directories for each employee, and a shared area for groups of developers and managers.

Now Mateo creates a set of policies to assign permissions to the users and user groups:

- **Home directory access for Zhang** – Mateo attaches a policy to Wei that lets him read, write, and list any objects with the Amazon S3 key prefix `/amzn-s3-demo-bucket/home/zhang/`
- **Home directory access for Major** – Mateo attaches a policy to Mary that lets her read, write, and list any objects with the Amazon S3 key prefix `/amzn-s3-demo-bucket/home/major/`

- **Shared directory access for the developers user group** – Mateo attaches a policy to the user group that lets developers read, write, and list any objects in `/amzn-s3-demo-bucket/share/developers/`
- **Shared directory access for the managers user group** – Mateo attaches a policy to the user group that lets managers read, write, and list objects in `/amzn-s3-demo-bucket/share/managers/`

> **Note**
>
> Amazon S3 doesn't automatically give a user who creates a bucket or object permission to perform other actions on that bucket or object. Therefore, in your IAM policies, you must explicitly give users permission to use the Amazon S3 resources they create.

For examples of what these policies might look like, see Access Control in the *Amazon Simple Storage Service User Guide*. For information on how policies are evaluated at runtime, see Policy evaluation logic.

User's job function change

At some point, one of the developers, Zhang Wei, changes job functions and becomes a manager. We assume that he no longer needs access to the documents in the `share/developers` directory. Mateo, as an admin, moves Wei to the `Managers` user group and out of the `Developers` user group. With just that simple reassignment, Wei automatically gets all permissions granted to the `Managers` user group, but can no longer access data in the `share/developers` directory.

Integration with a third-party business

Organizations often work with partner companies, consultants, and contractors. Example Corp has a partner called the Widget Company, and a Widget Company employee named Shirley Rodriguez needs to put data into a bucket for Example Corp's use. Nikki creates a user group called *WidgetCo* and a user named `Shirley` and adds Shirley to the WidgetCo user group. Nikki also creates a special bucket called *amzn-s3-demo-bucket1* for Shirley to use.

Nikki updates existing policies or adds new ones to accommodate the partner Widget Company. For example, Nikki can create a new policy that denies members of the WidgetCo user group the ability to use any actions other than write. This policy would be necessary only if there's a broad policy that gives all users access to a wide set of Amazon S3 actions.

IAM tutorials

The following tutorials present complete end-to-end procedures for common tasks for AWS Identity and Access Management (IAM). They are intended for a lab-type environment, with fictitious company names, user names, and so on. Their purpose is to provide general guidance. They are not intended for direct use in a production environment without careful review and adaptation to meet the unique needs of your organization's environment.

Tutorials

- IAM tutorial: Delegate access across AWS accounts using IAM roles
- IAM tutorial: Create and attach your first customer managed policy
- IAM tutorial: Define permissions to access AWS resources based on tags
- IAM tutorial: Permit users to manage their credentials and MFA settings
- IAM tutorial: Use a CloudFormation template to create a SAML Identity Provider (IdP)
- IAM tutorial: Use a CloudFormation template to create a SAML federated IAM role
- IAM tutorial: Use a CloudFormation template to create a SAML Identity Provider (IdP) and SAML federated IAM role

IAM tutorial: Delegate access across AWS accounts using IAM roles

> **Important**
>
> IAM best practices recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials. We recommend that you only use IAM users for specific use cases not supported by federated users.

This tutorial teaches you how to use a role to delegate access to resources in different AWS accounts called **Destination** and **Originating**. You share resources in one account with users in a different account. By setting up cross-account access in this way, you don't have to create individual IAM users in each account. In addition, users don't have to sign out of one account and sign in to

another account to access resources in different AWS accounts. After configuring the role, you see how to use the role from the AWS Management Console, the AWS CLI, and the API.

In this tutorial, the **Destination** account manages application data accessed by different applications and teams. In each account, you store application information in Amazon S3 buckets. You manage IAM users in the **Originating** account, where you have two IAM user roles: **Developers** and **Analysts**. Developers and Analysts use the **Originating** account to generate data shared by multiple microservices. Both roles have permissions to work in the Originating account and access resources there. From time to time, a developer must update the shared data in the **Destination** account. The developers store this data in an Amazon S3 bucket called `amzn-s3-demo-bucket-shared-container`.

At the end of this tutorial, you have the following:

- Users in the **Originating** account (the trusted account) allowed to assume a specific role in the **Destination** account.
- A role in the **Destination** account (the trusting account) allowed to access a specific Amazon S3 bucket.
- The `amzn-s3-demo-bucket-shared-container` bucket in the **Destination** account.

Developers can use the role in the AWS Management Console to access the `amzn-s3-demo-bucket-shared-container` bucket in the **Destination** account. They can also access the bucket by using API calls authenticated by temporary credentials provided by the role. Similar attempts by an Analyst to use the role fail.

This workflow has three basic steps:

Create a role in the Destination Account

First, you use the AWS Management Console to establish trust between the **Destination** account (ID number 999999999999) and the **Originating** account (ID number 111111111111). You start by creating an IAM role named *UpdateData*. When you create the role, you define the **Originating** account as a trusted entity and specify a permissions policy that allows trusted users to update the `amzn-s3-demo-bucket-shared-container` bucket.

Grant access to the role

In this section, you modify the role policy to deny Analysts access to the `UpdateData` role. Because Analysts have PowerUser access in this scenario, and you must explicitly *deny* the ability to use the role.

Test access by switching roles

Finally, as a Developer, you use the `UpdateData` role to update the `amzn-s3-demo-bucket-shared-container` bucket in the **Destination** account. You see how to access the role through the AWS console, the AWS CLI, and the API.

Considerations

Before you use IAM roles to delegate resource access across AWS accounts, it's important to consider the following:

- You cannot switch to a role when you sign in as the AWS account root user.
- IAM roles and resource-based policies delegate access across accounts only within a single partition. For example, assume that you have an account in US West (N. California) in the standard `aws` partition. You also have an account in China (Beijing) in the `aws-cn` partition. You can't use an Amazon S3 resource-based policy in your account in China (Beijing) to allow access for users in your standard `aws` account.
- You can use AWS IAM Identity Center to facilitate single sign-on (SSO) for external AWS accounts (accounts outside your AWS Organizations) using Security Assertion Markup Language (SAML). For details, see Integrate external AWS accounts into AWS IAM Identity Center for central access management with independent billing using SAML 2.0
- You can associate roles to AWS resources like Amazon EC2 instances or AWS Lambda functions. For details, see Create a role to delegate permissions to an AWS service.
- If you want to have an application assume a role in another AWS account, you can use the AWS SDK for cross account role assumption. For more information, see Authentication and access in the *AWS SDKs and Tools Reference Guide*.
- Switching roles using the AWS Management Console only works with accounts that do not require an `ExternalId`. For example, assume that you grant access to your account to a third party and require an `ExternalId` in a `Condition` element in your permissions policy. In that case, the third party can access your account only by using the AWS API or a command line tool. The third party cannot use the console because it must supply a value for `ExternalId`. For

more information about this scenario, see Access to AWS accounts owned by third parties, and How to enable cross account access to the AWS Management Console in the AWS Security Blog.

Prerequisites

This tutorial assumes that you have the following already in place:

- **Two** separate AWS accounts that you can use, one to represent the **Originating** account, and one to represent the **Destination** account.
- Users and roles in the **Originating** account created and configured as follows:

Job title	User	Permissions
Developer	David	Both users can sign in and use the AWS Management Console in the **Originating** account.
Analyst	Jane	

- You do not need to create any users in the **Destination** account.
- An Amazon S3 bucket created in the **Destination** account. You can call it `amzn-s3-demo-bucket-shared-container` in this tutorial, but because S3 bucket names must be globally unique, you must use a bucket with a different name.

Create a role in the Destination Account

You can allow users from one AWS account to access resources in another AWS account. In this tutorial, we'll do this by creating a role that defines who can access it and what permissions it grants to users that switch to it.

In this step of the tutorial, you create the role in the **Destination** account and specify the **Originating** account as a trusted entity. You also limit the role permissions to only read and write access to the `amzn-s3-demo-bucket-shared-container` bucket. Anyone granted permission to use the role can read and write to the `shared-container` bucket.

Before you can create a role, you need the *account ID* of the **Originating** AWS account. Each AWS account has a unique account ID identifier assigned to it.

To obtain the Originating AWS account ID

1. Sign in to the AWS Management Console as an administrator of the **Originating** account, and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console, choose your user name on the navigation bar in the upper right. It typically looks like this: ***username*@*account_ID_number_or_alias***.

   For this scenario, you can use the account ID 111111111111 for the **Originating** account. However, you should use a valid account ID if you use this scenario in your test environment.

To create a role in the Destination account that can be used by the Originating account

1. Sign in to the AWS Management Console as an administrator of the **Destination** account, and open the IAM console.
2. Before creating the role, prepare the managed policy that defines the permissions for the role requirements. You attach this policy to the role in a later step.

   You want to set read and write access to the `amzn-s3-demo-bucket-shared-container` bucket. Although AWS provides some Amazon S3 managed policies, there isn't one that provides read and write access to a single Amazon S3 bucket. You can create your own policy instead.

   In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Choose the **JSON** tab and copy the text from the following JSON policy document. Paste this text into the **JSON** text box, replacing the resource ARN (`arn:aws:s3:::shared-container`) with the real one for your Amazon S3 bucket.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    },
    {
```

```
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
       ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket-shared-container"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket-shared-container/*"
    }
  ]
}
```

The `ListAllMyBuckets` action grants permission to list all buckets owned by the authenticated sender of the request. The `ListBucket` permission allows users to view objects in the `amzn-s3-demo-bucket-shared-container` bucket. The `GetObject`, `PutObject`, `DeleteObject` permissions allows users to view, update, and delete contents in the `amzn-s3-demo-bucket-shared-container` bucket.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

4. On the **Review and create** page, type **`read-write-app-bucket`** for the policy name. Review the permissions granted by your policy, and then choose **Create policy** to save your work.

   The new policy appears in the list of managed policies.

5. In the navigation pane, choose **Roles** and then choose **Create role**.
6. Choose the **An AWS account** role type.
7. For **Account ID**, type the **Originating** account ID.

This tutorial uses the example account ID **111111111111** for the **Originating** account. You should use a valid account ID. If you use an invalid account ID, such as **111111111111**, IAM does not let you create the new role.

For now you do not need to require an external ID, or require users to have multi-factor authentication (MFA) in order to assume the role. Leave these options unselected. For more information, see AWS Multi-factor authentication in IAM.

8. Choose **Next: Permissions** to set the permissions associated with the role.
9. Select the checkbox next to the policy that you created previously.

> **Tip**
>
> For **Filter**, choose **Customer managed** to filter the list to include only the policies that you created. This hides the AWS created policies and makes it much easier to find the one you need.

Then, choose **Next**.

10. (Optional) Add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
11. (Optional) For **Description**, enter a description for the new role.
12. After reviewing the role, choose **Create role**.

The `UpdateData` role appears in the list of roles.

Now you must obtain the Amazon Resource Name (ARN) of the role, a unique identifier for the role. When you modify the Developer's role in the Originating account, you specify the role ARN from the Destination account to grant or deny permissions.

To obtain the ARN for UpdateData

1. In the navigation pane of the IAM console, choose **Roles**.
2. In the list of roles, choose the `UpdateData` role.
3. In the **Summary** section of the details pane, copy the **Role ARN** value.

The Destination account has an account ID of 999999999999, so the role ARN is `arn:aws:iam::999999999999:role/UpdateData`. Ensure that you provide the real AWS account ID for the Destination account.

At this point, you have established trust between the **Destination** and **Originating** accounts. You did this by creating a role in the **Destination** account that identifies the **Originating** account as a trusted principal. You also defined what the users who switch to the `UpdateData` role can do.

Next, modify the permissions for the Developer role.

Grant access to the role

At this point, both Analysts and Developers have permissions that allow them to manage data in the **Originating** account. Use the following required steps for adding permissions to allow switching to the role.

To modify the Developers role to allow them to switch to the UpdateData role

1. Sign in as an administrator in the **Originating** account, and open the IAM console.
2. Choose **Roles**, and then choose **Developers**.
3. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Create inline policy**.
4. Choose the **JSON** tab.
5. Add the following policy statement to allow the `AssumeRole` action on the `UpdateData` role in the Destination account. Be sure that you change *`DESTINATION-ACCOUNT-ID`* in the `Resource` element to the actual AWS account ID of the Destination account.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::111122223333:role/UpdateData"
    }
}
```

The `Allow` effect explicitly allows the Developers group access to the `UpdateData` role in the Destination account. Any developer who tries to access the role succeeds.

6. Choose **Review policy**.
7. Type a **Name** such as **`allow-assume-S3-role-in-destination`**.
8. Choose **Create policy**.

In most environments, you may not need the following procedure. If, however, you use PowerUserAccess permissions, then some groups might already be able to switch roles. The following procedure shows how to add a `"Deny"` permission to the Analysts group to ensure that they cannot assume the role. If you do not need this procedure in your environment, then we recommend that you do not add it. `"Deny"` permissions make the overall permissions picture more complicated to manage and understand. Use `"Deny"` permissions only when you do not have a better option.

To modify the Analysts role to deny permission to assume the `UpdateData` role

1. Choose **Roles**, and then choose **Analysts**.
2. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Create inline policy**.
3. Choose the **JSON** tab.
4. Add the following policy statement to deny the `AssumeRole` action on the `UpdateData` role. Be sure that you change *`DESTINATION-ACCOUNT-ID`* in the `Resource` element to the actual AWS account ID of the Destination account.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::111122223333:role/UpdateData"
    }
}
```

The `Deny` effect explicitly denies the Analysts group access to the `UpdateData` role in the Destination account. Any analyst who tries to access the role receives an access denied message.

5. Choose **Review policy**.
6. Type a **Name** like **`deny-assume-S3-role-in-destination`**.
7. Choose **Create policy**.

The Developers role now has permissions to use the `UpdateData` role in the Destination account. The Analysts role is prevented from using the `UpdateData` role.

Next, you can see how David, a developer, can access the `amzn-s3-demo-bucket-shared-container` bucket in the Destination account. David can access the bucket from the AWS Management Console, the AWS CLI, or the AWS API.

Test access by switching roles

After completing the first two steps of this tutorial, you have a role that grants access to a resource in the **Destination** account. You also have one role in the **Originating** account with users allowed to use that role. This step discusses how to test switching to that role from the AWS Management Console, the AWS CLI, and the AWS API.

To get help with common issues that you might encounter when working with IAM roles, see Troubleshoot IAM roles.

Switch roles (console)

If David needs to update data in the **Destination** account in the AWS Management Console, he can do so by using **Switch Role**. He specifies the account ID or alias and the role name, and his permissions immediately switch to those permitted by the role. He can then use the console to work with the `amzn-s3-demo-bucket-shared-container` bucket, but cannot work with any other resources in **Destination**. While David uses the role, he also cannot make use of his power-user privileges in the **Originating** account. That's because only one set of permissions can be in effect at a time.

IAM provides two ways that David can use to enter the **Switch Role** page:

- David receives a link from their administrator that points to a predefined Switch Role configuration. The link is provided to the administrator on the final page of the **Create role** wizard or on the **Role Summary** page for a cross-account role. Choosing this link takes David to the **Switch Role** page with the **Account ID** and **Role name** fields already filled in. All David needs to do is choose **Switch Roles**.
- The administrator does not send the link in email, but instead sends the **Account ID** number and **Role Name** values. To switch roles, David must manually enter the values. This is illustrated in the following procedure.

To assume a role

1. David signs into the AWS Management Console using his normal user in the **Originating** account.
2. They choose the link that the administrator emailed to them. This takes David to the **Switch Role** page with the account ID or alias and the role name information already filled in.

   —or—

   David chooses their name (the Identity menu) on the navigation bar, and then chooses **Switch Roles**.

   If this is the first time that David tries to access the Switch Role page this way, he first lands on a first-run **Switch Role** page. This page provides additional information on how switching roles can permit users to manage resources across AWS accounts. David must choose **Switch Role** on this page to complete the rest of this procedure.
3. Next, in order to access the role, David must manually type the Destination account ID number (999999999999) and the role name (`UpdateData`).

   Also, David wants to monitor which roles and associated permissions currently active in IAM. To keep track of this information, he types `Destination` in the **Display Name** text box, chooses the red color option, and then chooses **Switch Role**.
4. David can now use the Amazon S3 console to work with the Amazon S3 bucket, or any other resource to which the `UpdateData` role has permissions.
5. When done, David can return to their original permissions. To do that, they choose the **Destination** role display name on the navigation bar and then choose **Back to David @ 111111111111**.

6. The next time that David wants to switch roles and chooses the **Identity** menu in the navigation bar, he sees the Destination entry still there from last time. He can simply choose that entry to switch roles immediately without reentering the account ID and role name.

Switch roles (AWS CLI)

If David needs to work in the **Destination** environment at the command line, he can do so by using the AWS CLI. He runs the `aws sts assume-role` command and passes the role ARN to get temporary security credentials for that role. He then configures those credentials in environment variables so subsequent AWS CLI commands work using the role's permissions. While David uses the role, he cannot use his power-user privileges in the **Originating** account, because only one set of permissions can be in effect at a time.

Note that all access keys and tokens are examples only and cannot be used as shown. Replace with the appropriate values from your live environment.

To assume a role

1. David opens a command prompt window, and confirms that the AWS CLI client is working by running the command:

```
aws help
```

> **Note**
>
> David's default environment uses the `David` user credentials from his default profile that he created with the `aws configure` command. For more information, see Configuring the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

2. He begins the switch role process by running the following command to switch to the `UpdateData` role in the **Destination** account. He received the role ARN from the administrator that created the role. The command requires that you provide a session name as well, you can choose any text you like for that.

```
aws sts assume-role --role-arn "arn:aws:iam::999999999999:role/UpdateData" --role-
session-name "David-ProdUpdate"
```

David then sees the following in the output:

```
{
    "Credentials": {
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": "AQoDYXdzEGcaEXAMPLE2gsYULo
+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLE
CvSRyh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/
uZEXAMPLECihzFB5lTYLto9dyBgSDy
EXAMPLE9/
g7QRUhZp4bqbEXAMPLENwGPyOj59pFA4lNKCIkVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3Uuysg
sKdEXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP
+4eZScEXAMPLEsnf87e
NhyDHq6ikBQ==",
        "Expiration": "2014-12-11T23:08:07Z",
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE"
    }
}
```

3. David sees the three pieces that they need in the Credentials section of the output.

   - `AccessKeyId`
   - `SecretAccessKey`
   - `SessionToken`

   David needs to configure the AWS CLI environment to use these parameters in subsequent calls. For information about the various ways to configure your credentials, see Configuring the AWS Command Line Interface. You cannot use the `aws configure` command because it does not support capturing the session token. However, you can manually enter the information into a configuration file. Because these are temporary credentials with a relatively short expiration time, it is easiest to add them to the environment of your current command line session.

4. To add the three values to the environment, David cuts and pastes the output of the previous step into the following commands. You might want to cut and paste into a simple text editor to address line wrap issues in the output of the session token. It must be added as a single long string, even though it is shown line wrapped here for clarity.

The following example shows commands given in the Windows environment, where "set" is the command to create an environment variable. On a Linux or macOS computer, you would use the command "export" instead. All other parts of the example are valid in all three environments.

For details on using Tools for Windows Powershell, see Switch to an IAM role (Tools for Windows PowerShell)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEGcaEXAMPLE2gsYULo
+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLECvS
Ryh0FW7jEXAMPLEW+vE/7s1HRpXviG7b+qYf4nD00EXAMPLEmj4wxS04L/
uZEXAMPLECihzFB5lTYLto9dyBgSDyEXA
MPLEKEY9/
g7QRUhZp4bqbEXAMPLENwGPyOj59pFA4lNKCIkVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3UusKd
EXAMPLE1TVastU1A0SKFEXAMPLEiywCC/Cs8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP
+4eZScEXAMPLENhykxiHen
DHq6ikBQ==
```

At this point, any following commands run under the permissions of the role identified by those credentials. In David's case, the `UpdateData` role.

> **Important**
>
> You can save your frequently used configuration settings and credentials in files that are maintained by the AWS CLI. For more information, see Using existing configuration and credentials files in the *AWS Command Line Interface User Guide*.

5. Run the command to access the resources in the Destination account. In this example, David lists the contents of their S3 bucket with the following command.

```
aws s3 ls s3://shared-container
```

Because Amazon S3 bucket names are universally unique, there is no need to specify the account ID that owns the bucket. To access resources for other AWS services, refer to the AWS CLI documentation for that service for the commands and syntax required to reference its resources.

Using AssumeRole (AWS API)

When David needs to make an update to the **Destination** account from code, he makes an `AssumeRole` call to assume the `UpdateData` role. The call returns temporary credentials that he can use to access the `amzn-s3-demo-bucket-shared-container` bucket in the **Destination** account. With those credentials, David can make API calls to update the `amzn-s3-demo-bucket-shared-container` bucket. However, he cannot make API calls to access any other resources in the **Destination** account, even though he has power-user permissions in the **Originating** account.

To assume a role

1. David calls `AssumeRole` as part of an application. They must specify the `UpdateData` ARN: `arn:aws:iam::999999999999:role/UpdateData`.

   The response from the `AssumeRole` call includes the temporary credentials with an `AccessKeyId` and a `SecretAccessKey`. It also includes an `Expiration` time that indicates when the credentials expire and you must request new ones. When you set up role chaining with the AWS SDK, many credential providers automatically refresh credentials before they expire.
2. With the temporary credentials, David makes an `s3:PutObject` call to update the `amzn-s3-demo-bucket-shared-container` bucket. They would pass the credentials to the API call as the `AuthParams` parameter. Because the temporary role credentials have only read and write access to the `amzn-s3-demo-bucket-shared-container` bucket, any other actions in the Destination account are denied.

For a code example (using Python), see Switch to an IAM role (AWS API).

Additional resources

The following resources can help you learn more about topics in this tutorial:

- For more information about IAM users, see IAM Identities.
- For more information about Amazon S3 buckets, see Create a Bucket in the *Amazon Simple Storage Service User Guide*.
- To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

Summary

You have completed the cross-account API access tutorial. You created a role to establish trust with another account and defined what actions trusted entities can take. Then, you modified a role policy to control which IAM users can access the role. As a result, developers from the **Originating** account can make updates to the `amzn-s3-demo-bucket-shared-container` bucket in the **Destination** account by using temporary credentials.

IAM tutorial: Create and attach your first customer managed policy

In this tutorial, you use the AWS Management Console to create a customer managed policy and then attach that policy to an IAM user in your AWS account. The policy you create allows an IAM test user to sign in directly to the AWS Management Console with read-only permissions.

This workflow has three basic steps:

Step 1: Create the policy

By default, IAM users do not have permissions to do anything. They cannot access the AWS Management Console or manage the data within unless you allow it. In this step, you create a customer managed policy that allows any attached user to sign in to the console.

Step 2: Attach the policy

When you attach a policy to a user, the user inherits all of the access permissions that are associated with that policy. In this step, you attach the new policy to a test user.

Step 3: Test user access

Once the policy is attached, you can sign in as the user and test the policy.

Prerequisites

To perform the steps in this tutorial, you need to already have the following:

- An AWS account that you can sign in to as an IAM user with administrative permissions.
- A test IAM user that has no permissions assigned or group memberships as follows:

User name	Group	Permissions
PolicyUser	<none>	<none>

Step 1: Create the policy

In this step, you create a customer managed policy that allows any attached user to sign in to the AWS Management Console with read-only access to IAM data.

To create the policy for your test user

1. Sign in to the IAM console at https://console.aws.amazon.com/iam/ with your user that has administrator permissions.
2. In the navigation pane, choose **Policies**.
3. In the content pane, choose **Create policy**.
4. Choose the **JSON** option and copy the text from the following JSON policy document. Paste this text into the **JSON** text box.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [ {
        "Effect": "Allow",
        "Action": [
            "iam:GenerateCredentialReport",
            "iam:Get*",
            "iam:List*"
        ],
        "Resource": "*"
    } ]
}
```

5. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Review policy** in the **Visual** editor tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

6. On the **Review and create** page, type **`UsersReadOnlyAccessToIAMConsole`** for the policy name. Review the permissions granted by your policy, and then choose **Create policy** to save your work.

   The new policy appears in the list of managed policies and is ready to attach.

Step 2: Attach the policy

Next you attach the policy you just created to your test IAM user.

To attach the policy to your test user

1. In the IAM console, in the navigation pane, choose **Policies**.
2. At the top of the policy list, in the search box, start typing **`UsersReadOnlyAccessToIAMConsole`** until you can see your policy. Then choose the radio button next to **UsersReadOnlyAccessToIAMConsole** in the list.
3. Choose the **Actions** button, and then choose **Attach**.
4. In IAM entities choose the option to filter for **Users**.
5. In the search box, start typing **`PolicyUser`** until that user is visible on the list. Then check the box next to that user in the list.
6. Choose **Attach policy**.

You have attached the policy to your IAM test user, which means that user now has read-only access to the IAM console.

Step 3: Test user access

For this tutorial, we recommend that you test access by signing in as the test user so you can see what your users might experience.

To test access by signing in with your test user

1. Sign in to the IAM console at https://console.aws.amazon.com/iam/ with your `PolicyUser` test user.
2. Browse through the pages of the console and try to create a new user or group. Notice that `PolicyUser` can display data but cannot create or modify existing IAM data.

Related resources

For related information, see the following resources:

- Managed policies and inline policies
- Control IAM user access to the AWS Management Console

Summary

You've now successfully completed all of the steps necessary to create and attach a customer managed policy. As a result, you are able to sign in to the IAM console with your test account to see what the experience is like for your users.

IAM tutorial: Define permissions to access AWS resources based on tags

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM resources, including IAM entities (users or roles) and to AWS resources. You can define policies that use tag condition keys to grant permissions to your principals based on their tags. When you use tags to control access to your AWS resources, you allow your teams and resources to grow with fewer changes to AWS policies. ABAC policies are more flexible than traditional AWS policies, which require you to list each individual resource. For more information about ABAC and its advantage over traditional policies, see Define permissions based on attributes with ABAC authorization.

> **Note**
>
> You must pass a single value for each session tag. AWS Security Token Service does not support multi-valued session tags.

Topics

- Tutorial overview
- Prerequisites
- Step 1: Create test users
- Step 2: Create the ABAC policy
- Step 3: Create roles
- Step 4: Test creating secrets
- Step 5: Test viewing secrets
- Step 6: Test scalability
- Step 7: Test updating and deleting secrets
- Summary
- Related resources
- IAM tutorial: Use SAML session tags for ABAC

Tutorial overview

This tutorial shows how to create and test a policy that allows IAM roles with principal tags to access resources with matching tags. When a principal makes a request to AWS, their permissions are granted based on whether the principal and resource tags match. This strategy allows individuals to view or edit only the AWS resources required for their jobs.

Scenario

Assume that you're a lead developer at a large company named Example Corporation, and you're an experienced IAM administrator. You're familiar with creating and managing IAM users, roles, and policies. You want to ensure that your development engineers and quality assurance team members can access the resources they need. You also need a strategy that scales as your company grows.

You choose to use AWS resource tags and IAM role principal tags to implement an ABAC strategy for services that support it, beginning with AWS Secrets Manager. To learn which services support authorization based on tags, see AWS services that work with IAM. To learn which tagging condition keys you can use in a policy with each service's actions and resources, see Actions, Resources, and Condition Keys for AWS Services. You can configure your SAML-based or web

identity provider to pass session tags to AWS. When your employees federate into AWS, their attributes are applied to their resulting principal in AWS. You can then use ABAC to allow or deny permissions based on those attributes. To learn how using session tags with a SAML federated identity differs from this tutorial, see IAM tutorial: Use SAML session tags for ABAC.

Your Engineering and Quality Assurance team members are on either the **Pegasus** or **Unicorn** project. You choose the following 3-character project and team tag values:

- `access-project` = `peg` for the **Pegasus** project
- `access-project` = `uni` for the **Unicorn** project
- `access-team` = `eng` for the Engineering team
- `access-team` = `qas` for the Quality Assurance team

Additionally, you choose to require the `cost-center` cost allocation tag to enable custom AWS billing reports. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Summary of key decisions

- Employees sign in with IAM user credentials and then assume the IAM role for their team and project. If your company has its own identity system, you can set up federation to allow employees to assume a role without IAM users. For more information, see IAM tutorial: Use SAML session tags for ABAC.
- The same policy is attached to all of the roles. Actions are allowed or denied based on tags.
- Employees can create new resources, but only if they attach the same tags to the resource that are applied to their role. This ensures that employees can view the resource after they create it. Administrators are no longer required to update policies with the ARN of new resources.
- Employees can read resources owned by their team, regardless of the project.
- Employees can update and delete resources owned by their own team and project.
- IAM administrators can add a new role for new projects. They can create and tag a new IAM user to allow access to the appropriate role. Administrators are not required to edit a policy to support a new project or team member.

In this tutorial, you will tag each resource, tag your project roles, and add policies to the roles to allow the behavior previously described. The resulting policy allows the roles `Create`, `Read`,

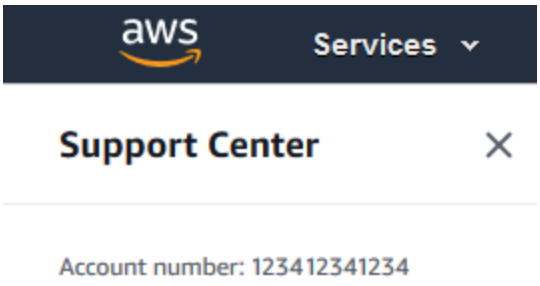
`Update`, and `Delete` access to resources that are tagged with the same project and team tags. The policy also allows cross-project `Read` access for resources that are tagged with the same team. The diagram shows two projects where roles are limited to read only access outside of their project while have permissions to create, read, update, and delete resources in their own project.

Prerequisites

To perform the steps in this tutorial, you must already have the following:

- An AWS account that you can sign in to as a user with administrative permissions.
- Your 12-digit account ID, which you use to create the roles in step 3.

  To find your AWS account ID number using the AWS Management Console, choose **Support** on the navigation bar on the upper right, and then choose **Support Center**. The account number (ID) appears in the navigation pane on the left.



- Experience creating and editing IAM users, roles, and policies in the AWS Management Console. However, if you need help remembering an IAM management process, this tutorial provides links where you can view step-by-step instructions.

Step 1: Create test users

For testing, create four IAM users with permissions to assume roles with the same tags. This makes it easier to add more users to your teams. When you tag the users, they automatically get access to assume the correct role. You don't have to add the users to the trust policy of the role if they work on only one project and team.

1. Create the following customer managed policy named `access-assume-role`. For more information about creating a JSON policy, see Creating IAM policies.

   ABAC policy: Assume any ABAC role, but only when the user and role tags match

The following policy allows a user to assume any role in your account with the `access-` name prefix. The role must also be tagged with the same project, team, and cost center tags as the user.

To use this policy, replace the *italicized placeholder text* with your account information.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "TutorialAssumeRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::111122223333:role/access-*",
            "Condition": {
                "StringEquals": {
                    "iam:ResourceTag/access-project": "${aws:PrincipalTag/
access-project}",
                    "iam:ResourceTag/access-team": "${aws:PrincipalTag/
access-team}",
                    "iam:ResourceTag/cost-center": "${aws:PrincipalTag/cost-
center}"
                }
            }
        }
    ]
}
```

To scale this tutorial to a large number of users, you can attach the policy to a group and add each user to the group. For more information, see Create IAM groups and Edit users in IAM groups.

2. Create the following IAM users, attach the `access-assume-role` permissions policy. Make sure you select **Provide user access to the AWS Management Console**, and then add the following tags.

User name	User tag key	User tag value
access-Arnav-peg-eng	access-project	peg
	access-team	eng
	cost-center	987654
access-Mary-peg-qas	access-project	peg
	access-team	qas
	cost-center	987654
access-Saanvi-uni-eng	access-project	uni
	access-team	eng
	cost-center	123456
access-Carlos-uni-qas	access-project	uni
	access-team	qas
	cost-center	123456

Step 2: Create the ABAC policy

Create the following policy named **access-same-project-team**. You will add this policy to the roles in a later step. For more information about creating a JSON policy, see Creating IAM policies.

For additional policies that you can adapt for this tutorial, see the following pages:

- Controlling access for IAM principals
- Amazon EC2: Allows starting or stopping EC2 instances a user has tagged, programmatically and in the console
- EC2: Start or stop instances based on matching principal and resource tags
- EC2: Start or stop instances based on tags

- IAM: Assume roles that have a specific tag

ABAC Policy: Access Secrets Manager Resources Only When the Principal and Resource Tags Match

The following policy allows principals to create, read, edit, and delete resources, but only when those resources are tagged with the same key-value pairs as the principal. When a principal creates a resource, they must add `access-project`, `access-team`, and `cost-center` tags with values that match the principal's tags. The policy also allows adding optional `Name` or `OwnedBy` tags.

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
     {
         "Sid": "AllActionsSecretsManagerSameProjectSameTeam",
         "Effect": "Allow",
         "Action": "secretsmanager:*",
         "Resource": "*",
         "Condition": {
             "StringEquals": {
                 "aws:ResourceTag/access-project": "${aws:PrincipalTag/access-
project}",
                 "aws:ResourceTag/access-team": "${aws:PrincipalTag/access-
team}",
                 "aws:ResourceTag/cost-center": "${aws:PrincipalTag/cost-center}"
             },
             "ForAllValues:StringEquals": {
                 "aws:TagKeys": [
                     "access-project",
                     "access-team",
                     "cost-center",
                     "Name",
                     "OwnedBy"
                 ]
             },
             "StringEqualsIfExists": {
                 "aws:RequestTag/access-project": "${aws:PrincipalTag/access-
project}",
                 "aws:RequestTag/access-team": "${aws:PrincipalTag/access-team}",
```

```
                "aws:RequestTag/cost-center": "${aws:PrincipalTag/cost-center}"
            }
        }
    },
    {
        "Sid": "AllResourcesSecretsManagerNoTags",
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetRandomPassword",
            "secretsmanager:ListSecrets"
        ],
        "Resource": "*"
    },
    {
        "Sid": "ReadSecretsManagerSameTeam",
        "Effect": "Allow",
        "Action": [
            "secretsmanager:Describe*",
            "secretsmanager:Get*",
            "secretsmanager:List*"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:ResourceTag/access-team": "${aws:PrincipalTag/access-team}"
            }
        }
    },
    {
        "Sid": "DenyUntagSecretsManagerReservedTags",
        "Effect": "Deny",
        "Action": "secretsmanager:UntagResource",
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringLike": {
                "aws:TagKeys": "access-*"
            }
        }
    },
    {
        "Sid": "DenyPermissionsManagement",
        "Effect": "Deny",
        "Action": "secretsmanager:*Policy",
        "Resource": "*"
```

```
        }
    ]
}
```

What does this policy do?

- The `AllActionsSecretsManagerSameProjectSameTeam` statement allows all of this service's actions on all related resources, but only if the resource tags match the principal tags. By adding `"Action": "secretsmanager:*"` to the policy, the policy grows as Secrets Manager grows. If Secrets Manager adds a new API operation, you are not required to add that action to the statement. The statement implements ABAC using three condition blocks. The request is allowed only if all three blocks return true.
  - The first condition block of this statement returns true if the specified tag keys are present on the resource, and their values match the principal's tags. This block returns false for mismatched tags, or for actions that don't support resource tagging. To learn which actions are not allowed by this block, see Actions, Resources, and Condition Keys for AWS Secrets Manager. That page shows that actions performed on the **Secret** resource type support the `secretsmanager:ResourceTag/tag-key` condition key. Some Secrets Manager actions don't support that resource type, including `GetRandomPassword` and `ListSecrets`. You must create additional statements to allow those actions.
  - The second condition block returns true if every tag key passed in the request is included in the specified list. This is done using `ForAllValues` with the `StringEquals` condition operator. If no keys or a subset of the set of keys is passed, then the condition returns true. This allows `Get*` operations that do not allow passing tags in the request. If the requester includes a tag key that is not in the list, the condition returns false. Every tag key that is passed in the request must match a member of this list. For more information, see Set operators for multivalued context keys.
  - The third condition block returns true if the request supports passing tags, if all three of the tags are present, and if they match the principal tag values. This block also returns true if the request does not support passing tags. This is thanks to `...IfExists` in the condition operator. The block returns false if there is no tag passed during an action that supports it, or if the tag keys and values don't match.
- The `AllResourcesSecretsManagerNoTags` statement allows the `GetRandomPassword` and `ListSecrets` actions that are not allowed by the first statement.

- The `ReadSecretsManagerSameTeam` statement allows read-only operations if the principal is tagged with the same access-team tag as the resource. This is allowed regardless of the project or cost-center tag.
- The `DenyUntagSecretsManagerReservedTags` statement denies requests to remove tags with keys that begin with "access-" from Secrets Manager. These tags are used to control access to resources, therefore removing tags might remove permissions.
- The `DenyPermissionsManagement` statement denies access to create, edit, or delete Secrets Manager resource-based policies. These policies could be used to change the permissions of the secret.

> **Important**
>
> This policy uses a strategy to allow all actions for a service, but explicitly deny permissions-altering actions. Denying an action overrides any other policy that allows the principal to perform that action. This can have unintended results. As a best practice, use explicit denies only when there is no circumstance that should allow that action. Otherwise, allow a list of individual actions, and the unwanted actions are denied by default.

Step 3: Create roles

Create the following IAM roles and attach the **`access-same-project-team`** policy that you created in the previous step. For more information about creating IAM roles, see Create a role to give permissions to an IAM user. If you choose to use federation instead of IAM users and roles, see IAM tutorial: Use SAML session tags for ABAC.

Job function	Role name	Role tags	Role description
Project Pegasus Engineering	access-peg-engineering	access-project = peg access-team = eng cost-center = 987654	Allows engineers to read all engineering resources and create and manage Pegasus engineering resources.

Job function	Role name	Role tags	Role description
Project Pegasus Quality Assurance	access-peg-quality-assurance	access-project = peg access-team = qas cost-center = 987654	Allows the QA team to read all QA resources and create and manage all Pegasus QA resources.
Project Unicorn Engineering	access-uni-engineering	access-project= uni access-team = eng cost-center = 123456	Allows engineers to read all engineering resources and create and manage Unicorn engineering resources.
Project Unicorn Quality Assurance	access-uni-quality-assurance	access-project = uni access-team = qas cost-center = 123456	Allows the QA team to read all QA resources and create and manage all Unicorn QA resources.

Step 4: Test creating secrets

The permissions policy attached to the roles allows the employees to create secrets. This is allowed only if the secret is tagged with their project, team, and cost center. Confirm that your permissions are working as expected by signing in as your users, assuming the correct role, and testing activity in Secrets Manager.

To test creating a secret with and without the required tags

1. In your main browser window, remain signed in as the administrator user so that you can review users, roles, and policies in IAM. Use a browser incognito window or separate browser for your testing. There, sign in as the `access-Arnav-peg-eng` IAM user and open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Attempt to switch to the `access-uni-engineering` role.

   This operation fails because the `access-project` and `cost-center` tag values do not match for the `access-Arnav-peg-eng` user and `access-uni-engineering` role.

   For more information about switching roles in the AWS Management Console, see Switch from a user to an IAM role (console)
3. Switch to the `access-peg-engineering` role.
4. Store a new secret using the following information. To learn how to store a secret, see Creating a Basic Secret in the *AWS Secrets Manager User Guide*.

   > **Important**
   >
   > Secrets Manager displays alerts that you don't have permissions for additional AWS services that work with Secrets Manager. For example, to create credentials for an Amazon RDS database, you must have permission to describe RDS instances, RDS clusters, and Amazon Redshift clusters. You can ignore these alerts since you aren't using these specific AWS services in this tutorial.

   1. In the **Select secret type** section, choose **Other type of secrets**. In the two text boxes, enter `test-access-key` and `test-access-secret`.
   2. Enter `test-access-peg-eng` for the **Secret name** field.
   3. Add different tag combinations from the following table and view the expected behavior.
   4. Choose **Store** to attempt to create the secret. When the storage fails, return to the previous Secrets Manager console pages and use the next tag set from the following table. The last tag set is allowed and will successfully create the secret.

   The following table shows ABAC tag combinations for the `test-access-peg-eng` role.

access-project Tag value	access-team Tag value	cost-center Tag value	Additional tags	Expected behavior
(none)	(none)	(none)	(none)	Denied because the access-project tag value does not match the role's value of peg.
uni	eng	987654	(none)	Denied because the access-project tag value does not match the role's value of peg.
peg	qas	987654	(none)	Denied because the access-team tag value does not match the role's value of eng.
peg	eng	123456	(none)	Denied because the cost-center tag value does not match the role's value of 987654.
peg	eng	987654	Owner = Jane	Denied because the additional tag owner is not allowed by the policy, even though all three required tags are present and their values match the role's values.
peg	eng	987654	Name = Jane	Allowed because all three required tags are present and their values match the role's values. You are also allowed to include the optional Name tag.

5. Sign out and repeat the first three steps of this procedure for each of the following roles and tag values. In the fourth step in this procedure, test any set of missing tags, optional tags, disallowed tags, and invalid tag values that you choose. Then use the required tags to create a secret with the following tags and name.

User name	Role name	Secret name	Secret tags
access-Mary-peg-qas	access-peg-quality-assurance	test-access-peg-qas	access-project = peg access-team = qas cost-center = 987654
access-Saanvi-uni-eng	access-uni-engineering	test-access-uni-eng	access-project = uni access-team = eng cost-center = 123456
access-Carlos-uni-qas	access-uni-quality-assurance	test-access-uni-qas	access-project = uni access-team = qas cost-center = 123456

Step 5: Test viewing secrets

The policy that you attached to each role allows the employees to view any secrets tagged with their team name, regardless of their project. Confirm that your permissions are working as expected by testing your roles in Secrets Manager.

To test viewing a secret with and without the required tags

1. Sign in as one of the following IAM users:
   - access-Arnav-peg-eng
   - access-Mary-peg-qas
   - access-Saanvi-uni-eng
   - access-Carlos-uni-qas

2. Switch to the matching role:

   - `access-peg-engineering`
   - `access-peg-quality-assurance`
   - `access-uni-engineering`
   - `access-uni-quality-assurance`

   For more information about switching roles in the AWS Management Console, see Switch from a user to an IAM role (console).

3. In the navigation pane on the left, choose the menu icon to expand the menu and then choose **Secrets**.
4. You should see all four secrets in the table, regardless of your current role. This is expected because the policy named `access-same-project-team` allows the `secretsmanager:ListSecrets` action for all resources.
5. Choose the name of one of the secrets.
6. On the details page for the secret, your role's tags determine whether you can view the page content. Compare the name of your role to the name of your secret. If they share the same team name, then the `access-team` tags match. If they don't match, then access is denied.

   The following table shows ABAC secret viewing behavior for each role.

Role name	Secret name	Expected behavior
`access-peg-engineering`	`test-access-peg-eng`	Allowed
	test-access-peg-qas	Denied
	test-access-uni-eng	Allowed
	test-access-uni-qas	Denied
access-peg-quality-assurance	test-access-peg-eng	Denied
	test-access-peg-qas	Allowed
	test-access-uni-eng	Denied

Role name	Secret name	Expected behavior
	test-access-uni-qas	Allowed
access-uni-engineering	test-access-peg-eng	Allowed
	test-access-peg-qas	Denied
	test-access-uni-eng	Allowed
	test-access-uni-qas	Denied
access-uni-quality-assurance	test-access-peg-eng	Denied
	test-access-peg-qas	Allowed
	test-access-uni-eng	Denied
	test-access-uni-qas	Allowed

7. From the breadcrumbs at the top of the page, choose **Secrets** to return to the list of secrets. Repeat the steps in this procedure using different roles to test whether you can view each of the secrets.

Step 6: Test scalability

An important reason for using attribute-based access control (ABAC) over role-based access control (RBAC) is scalability. As your company adds new projects, teams, or people to AWS, you don't need to update your ABAC-driven policies. For example, assume that Example Company is funding a new project, code named **Centaur**. An engineer named Saanvi Sarkar will be the lead engineer for **Centaur** while continuing to work on the **Unicorn** project. Saanvi will also review work for the **Peg** project. There are also several newly hired engineers, including Nikhil Jayashankar, who will work on only the **Centaur** project.

To add the new project to AWS

1. Sign in as the IAM administrator user and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane on the left, choose **Roles** and add an IAM role named `access-cen-engineering`. Attach the **`access-same-project-team`** permissions policy to the role and add the following role tags:

   - `access-project` = `cen`
   - `access-team` = `eng`
   - `cost-center` = `101010`

3. In the navigation pane on the left, choose **Users**.
4. Add a new user named `access-Nikhil-cen-eng`, attach the policy named `access-assume-role`, and add the following user tags.

   - `access-project` = `cen`
   - `access-team` = `eng`
   - `cost-center` = `101010`

5. Use the procedures in Step 4: Test creating secrets and Step 5: Test viewing secrets. In another browser window, test that Nikhil can create only **Centaur** engineering secrets, and that he can view all engineering secrets.
6. In the main browser window where you signed in as the administrator, choose the user `access-Saanvi-uni-eng`.
7. On the **Permissions** tab, remove the **access-assume-role** permissions policy.
8. Add the following inline policy named `access-assume-specific-roles`. For more information about adding an inline policy to a user, see To embed an inline policy for a user or role (console).

   ABAC policy: Assume only specific roles

   This policy allows Saanvi to assume the engineering roles for the **Pegasus** and **Centaur** projects. It is necessary to create this custom policy because IAM does not support multivalued tags. You can't tag Saanvi's user with `access-project` = `peg` and `access-project` = `cen`. Additionally, the AWS authorization model can't match both values. For more information, see Rules for tagging in IAM and AWS STS. Instead, you must manually specify the two roles that she can assume.

   To use this policy, replace the *italicized placeholder text* with your account information.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "TutorialAssumeSpecificRoles",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": [
                "arn:aws:iam::111122223333:role/access-peg-engineering",
                "arn:aws:iam::111122223333:role/access-cen-engineering"
            ]
        }
    ]
}
```

9. Use the procedures in Step 4: Test creating secrets and Step 5: Test viewing secrets. In another browser window, confirm that Saanvi can assume both roles. Check that she can create secrets for only her project, team, and cost center, depending on the role's tags. Also confirm that she can view details about any secrets owned by the engineering team, including the ones that she just created.

Step 7: Test updating and deleting secrets

The `access-same-project-team` policy that is attached to the roles allows the employees to update and delete any secrets tagged with their project, team, and cost center. Confirm that your permissions are working as expected by testing your roles in Secrets Manager.

To test updating and deleting a secret with and without the required tags

1. Sign in as one of the following IAM users:

   - `access-Arnav-peg-eng`
   - `access-Mary-peg-qas`
   - `access-Saanvi-uni-eng`
   - `access-Carlos-uni-qas`
   - `access-Nikhil-cen-eng`

2. Switch to the matching role:

   - `access-peg-engineering`
   - `access-peg-quality-assurance`
   - `access-uni-engineering`
   - `access-peg-quality-assurance`
   - `access-cen-engineering`

   For more information about switching roles in the AWS Management Console, see Switch from a user to an IAM role (console).

3. For each role, try to update the secret description and then try to delete the following secrets. For more information, see Modifying a Secret and Deleting and Restoring a Secret in the *AWS Secrets Manager User Guide*.

   The following table shows ABAC secret updating and deleting behavior for each role.

Role name	Secret name	Expected behavior
access-peg-engineering	test-access-peg-eng	Allowed
	test-access-uni-eng	Denied
	test-access-uni-qas	Denied
access-peg-quality-assurance	test-access-peg-qas	Allowed
	test-access-uni-eng	Denied
access-uni-engineering	test-access-uni-eng	Allowed
	test-access-uni-qas	Denied
access-peg-quality-assurance	test-access-uni-qas	Allowed

Summary

You've now successfully completed all of the steps necessary to use tags for attribute-based access control (ABAC). You've learned how to define a tagging strategy. You applied that strategy to your principals and resources. You created and applied a policy that enforces the strategy for Secrets Manager. You also learned that ABAC scales easily when you add new projects and team members. As a result, you are able to sign in to the IAM console with your test roles and experience how to use tags for ABAC in AWS.

> **Note**
>
> You added policies that allow actions only under specific conditions. If you apply a different policy to your users or roles that has broader permissions, then the actions might not be limited to require tagging. For example, if you give a user full administrative permissions using the `AdministratorAccess` AWS managed policy, then these policies don't restrict that access. For more information about how permissions are determined when multiple policies are involved, see How AWS enforcement code logic evaluates requests to allow or deny access.

Related resources

For related information, see the following resources:

- Define permissions based on attributes with ABAC authorization
- AWS global condition context keys
- Create a role to give permissions to an IAM user
- Tags for AWS Identity and Access Management resources
- Controlling access to AWS resources using tags
- Switch from a user to an IAM role (console)
- IAM tutorial: Use SAML session tags for ABAC

To learn how to monitor the tags in your account, see Monitor tag changes on AWS resources with serverless workflows and Amazon CloudWatch Events.

IAM tutorial: Use SAML session tags for ABAC

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called tags. You can attach tags to IAM resources, including IAM entities (users or roles), and to AWS resources. When the entities are used to make requests to AWS, they become principals and those principals include tags.

You can also pass session tags when you assume a role or federate a user. You can then define policies that use tag condition keys to grant permissions to your principals based on their tags. When you use tags to control access to your AWS resources, you allow your teams and resources to grow with fewer changes to AWS policies. ABAC policies are more flexible than traditional AWS policies, which require you to list each individual resource. For more information about ABAC and its advantage over traditional policies, see Define permissions based on attributes with ABAC authorization.

If your company uses a SAML-based identity provider (IdP) to manage corporate user identities, you can use SAML attributes for fine-grained access control in AWS. Attributes can include cost center identifiers, user email addresses, department classifications, and project assignments. When you pass these attributes as session tags, you can then control access to AWS based on these session tags.

To complete the ABAC tutorial by passing SAML attributes to your session principal, complete the tasks in IAM tutorial: Define permissions to access AWS resources based on tags, with the changes that are included in this topic.

Prerequisites

To perform the steps to use SAML session tags for ABAC, you must already have the following:

- Access to a SAML-based IdP where you can create test users with specific attributes.
- The ability to sign in as a user with administrative permissions.
- Experience creating and editing IAM users, roles, and policies in the AWS Management Console. However, if you need help remembering an IAM management process, the ABAC tutorial provides links where you can view step-by-step instructions.
- Experience setting up a SAML-based IdP in IAM. To view more details and links to detailed IAM documentation, see Passing session tags using AssumeRoleWithSAML.

Step 1: Create test users

Skip the instructions in Step 1: Create test users. Because your identities are defined in your provider, it's not necessary for you to add IAM users for your employees.

Step 2: Create the ABAC policy

Follow the instructions in Step 2: Create the ABAC policy to create the specified managed policy in IAM.

Step 3: Create and configure the SAML role

When you use the ABAC tutorial for SAML, you must perform additional steps to create the role, configure the SAML IdP, and enable AWS Management Console access. For more information, see Step 3: Create roles.

Step 3A: Create the SAML role

Create a single role that trusts your SAML identity provider and the `test-session-tags` user that you created in step 1. The ABAC tutorial uses separate roles with different role tags. Because you are passing session tags from your SAML IdP, you need only one role. To learn how to create a SAML-based role, see Create a role for SAML 2.0 federation (console).

Name the role `access-session-tags`. Attach the `access-same-project-team` permissions policy to the role. Edit the role trust policy to use the following policy. For detailed instructions on how to edit the trust relationship of a role, see Update a role trust policy.

The following role trust policy allows your SAML identity provider and the `test-session-tags` user to assume the role. When they assume the role, they must pass the three specified session tags. The `sts:TagSession` action is required to allow passing session tags.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSamlIdentityAssumeRole",
            "Effect": "Allow",
            "Action": [
```

```
                "sts:AssumeRoleWithSAML",
                "sts:TagSession"
            ],
            "Principal": {"Federated":"arn:aws:iam::123456789012:saml-
provider/ExampleCorpProvider"},
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/cost-center": "*",
                    "aws:RequestTag/access-project": "*",
                    "aws:RequestTag/access-team": [
                        "eng",
                        "qas"
                    ]
                },
                "StringEquals": {"SAML:aud": "https://region-
code.signin.aws.amazon.com/saml"}
            }
        }
    ]
}
```

The `AllowSamlIdentityAssumeRole` statement allows members of the Engineering and Quality Assurance teams to assume this role when they federate into AWS from the Example Corporation IdP. The `ExampleCorpProvider` SAML provider is defined in IAM. The administrator has already set up the SAML assertion to pass the three required session tags. The assertion can pass additional tags, but these three must be present. The identity's attributes can have any value for the `cost-center` and `access-project` tags. However, the `access-team` attribute value must match `eng` or `qas` to indicate that the identity is on the Engineering or Quality Assurance team.

Step 3B: Configure the SAML IdP

Configure your SAML IdP to pass the `cost-center`, `access-project`, and `access-team` attributes as session tags. For more information, see Passing session tags using AssumeRoleWithSAML.

To pass these attributes as session tags, include the following elements in your SAML assertion.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:cost-center">
  <AttributeValue>987654</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:access-project">
```

```
  <AttributeValue>peg</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:access-team">
  <AttributeValue>eng</AttributeValue>
</Attribute>
```

Step 3C: Enable console access

Enable console access for your federated SAML users. For more information, see Enabling SAML 2.0 federated principals to access the AWS Management Console.

Step 4: Test creating secrets

Federate into the AWS Management Console using the `access-session-tags` role. For more information, see Enabling SAML 2.0 federated principals to access the AWS Management Console. Then follow the instructions in Step 4: Test creating secrets to create secrets. Use different SAML identities with attributes to match the tags that are indicated in the ABAC tutorial. For more information, see Step 4: Test creating secrets.

Step 5: Test viewing secrets

Follow the instructions in Step 5: Test viewing secrets to view the secrets that you created in the previous step. Use different SAML identities with attributes to match the tags that are indicated in the ABAC tutorial.

Step 6: Test scalability

Follow the instructions in Step 6: Test scalability to test scalability. Do this by adding a new identity in your SAML-based IdP with the following attributes:

- `cost-center = 101010`
- `access-project = cen`
- `access-team = eng`

Step 7: Test updating and deleting secrets

Follow the instructions in Step 7: Test updating and deleting secrets to update and delete secrets. Use different SAML identities with attributes to match the tags that are indicated in the ABAC tutorial.

Important

Delete all of the secrets that you created to avoid billing charges. For details about pricing in Secrets Manager, see AWS Secrets Manager Pricing.

Summary

You've now successfully completed all of the steps necessary to use SAML session tags and resource tags for permissions management.

Note

You added policies that allow actions only under specific conditions. If you apply a different policy to your users or roles that has broader permissions, then the actions might not be limited to require tagging. For example, if you give a user full administrative permissions using the `AdministratorAccess` AWS managed policy, then these policies don't restrict that access. For more information about how permissions are determined when multiple policies are involved, see How AWS enforcement code logic evaluates requests to allow or deny access.

IAM tutorial: Permit users to manage their credentials and MFA settings

You can permit your users to manage their own multi-factor authentication (MFA) devices and credentials on the **Security credentials** page. You can use the AWS Management Console to configure credentials (access keys, passwords, signing certificates, and SSH public keys), delete or deactivate credentials that are not needed, and enable MFA devices for your users. This is useful for a small number of users, but that task could quickly become time consuming as the number of users grows. This tutorial shows you how to enable these best practices without burdening your administrators.

This tutorial shows how to allow users to access AWS services, but **only** when they sign in with MFA. If they are not signed in with an MFA device, then users cannot access other services.

This workflow has three basic steps.

Step 1: Create a policy to enforce MFA sign-in

Create a customer managed policy that prohibits all actions ***except*** the few IAM actions. These exceptions allow a user to change their own credentials and manage their MFA devices on the **Security credentials** page. For more information about accessing that page, see How IAM users change their own password (console).

Step 2: Attach policies to your test user group

Create a user group whose members have full access to all Amazon EC2 actions if they sign in with MFA. To create such a user group, you attach both the AWS managed policy called `AmazonEC2FullAccess` and the customer managed policy you created in the first step.

Step 3: Test your user's access

Sign in as the test user to verify that access to Amazon EC2 is blocked *until* the user creates an MFA device. The user can then sign in using that device.

Prerequisites

To perform the steps in this tutorial, you must already have the following:

- An AWS account that you can sign in to as an IAM user with administrative permissions.
- Your account ID number, which you type into the policy in Step 1.

  To find your account ID number, on the navigation bar at the top of the page, choose **Support** and then choose **Support Center**. You can find your account ID under this page's **Support** menu.
- A virtual (software-based) MFA device, FIDO security key, or hardware-based MFA device.
- A test IAM user who is a member of a user group as follows:

User name	User name instructions	User group name	Add user as a member	User group instructions
MFAUser	Choose only the option for **Enable console access – *optional***, and assign a password.	EC2MFA	MFAUser	Do NOT attach any policies or otherwise grant permissions to this user group.

Step 1: Create a policy to enforce MFA sign-in

You begin by creating an IAM customer managed policy that denies all permissions except those required for IAM users to manage their own credentials and MFA devices.

1. Sign in to the AWS Management Console as a user with administrator credentials. To adhere to IAM best practices, don't sign in with your AWS account root user credentials.

   > **Important**
   >
   > IAM best practices recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials. We recommend that you only use IAM users for specific use cases not supported by federated users.

2. Open the IAM console at https://console.aws.amazon.com/iam/.

3. In the navigation pane, choose **Policies**, and then choose **Create policy**.

4. Choose the **JSON** tab and copy the text from the following JSON policy document: AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page.

5. Paste the policy text into the **JSON** text box. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

   > **Note**
   >
   > You can switch between the **Visual editor** and **JSON** options anytime. However, the policy above includes the `NotAction` element, which is not supported in the visual editor. For this policy, you will see a notification on the **Visual editor** tab. Return to **JSON** to continue working with this policy.
   > This example policy does not allow users to reset a password while signing in to the AWS Management Console for the first time. We recommend that you do not grant permissions to new users until after they sign in and reset their password.

6. On the **Review and create** page, type **`Force_MFA`** for the policy name. For the policy description, type **`This policy allows users to manage their own passwords and MFA devices but nothing else unless they authenticate with MFA.`** In the

Tags area, you can optionally add tag key-value pairs to the customer managed policy. Review the permissions granted by your policy, and then choose **Create policy** to save your work.

The new policy appears in the list of managed policies and is ready to attach.

Step 2: Attach policies to your test user group

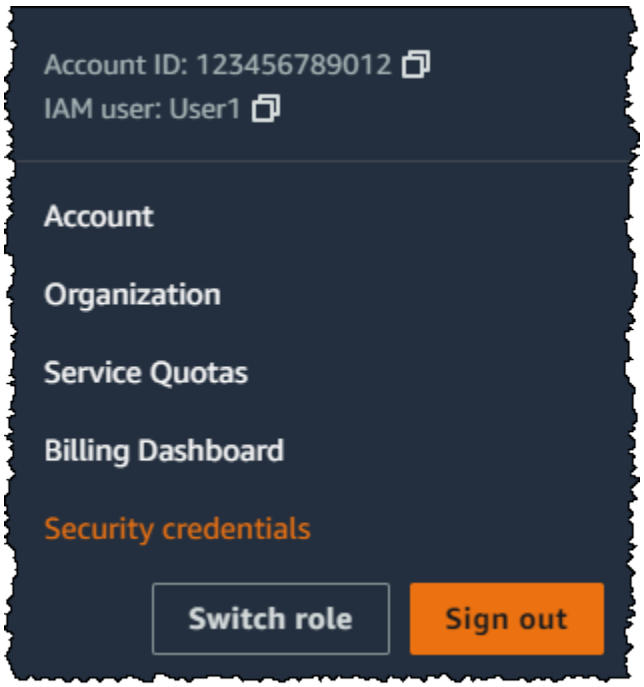
Next you attach two policies to the test IAM user group, which will be used to grant the MFA-protected permissions.

1. In the navigation pane, choose **User groups**.
2. In the search box, type **EC2MFA**, and then choose the group name (not the checkbox) in the list.
3. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Attach policies**.
4. On the **Attach permission policies to EC2MFA group** page, in the search box, type **EC2Full**. Then select the checkbox next to **AmazonEC2FullAccess** in the list. Don't save your changes yet.
5. In the search box, type **Force**, and then select the checkbox next to **Force_MFA** in the list.
6. Choose **Attach policies**.

Step 3: Test your user's access

In this part of the tutorial, you sign in as the test user and verify that the policy works as intended.

1. Sign in to your AWS account as **MFAUser** with the password you assigned in the previous section. Use the URL: `https://<alias or account ID number>.signin.aws.amazon.com/console`
2. Choose **EC2** to open the Amazon EC2 console and verify that the user has no permissions to do anything.
3. In the navigation bar on the upper right, choose the `MFAUser` user name, and then choose **Security credentials**.



4. Now add an MFA device. In the **Multi-factor Authentication (MFA)** section, choose **Assign MFA device**.

   > **Note**
   >
   > You might receive an error that you are not authorized to perform `iam:DeleteVirtualMFADevice`. This could happen if someone previously began assigning a virtual MFA device to this user and cancelled the process. To continue, you or another administrator must delete the user's existing unassigned virtual MFA device. For more information, see I am not authorized to perform: iam:DeleteVirtualMFADevice.

5. For this tutorial, we use a virtual (software-based) MFA device, such as the Google Authenticator app on a mobile phone. Choose **Authenticator app**, and then click **Next**.

   IAM generates and displays configuration information for the virtual MFA device, including a QR code graphic. The graphic is a representation of the secret configuration key that is available for manual entry on devices that do not support QR codes.

6. Open your virtual MFA app. (For a list of apps that you can use for hosting virtual MFA devices, see Virtual MFA Applications.) If the virtual MFA app supports multiple accounts (multiple virtual MFA devices), choose the option to create a new account (a new virtual MFA device).

7. Determine whether the MFA app supports QR codes, and then do one of the following:

- From the wizard, choose **Show QR code**. Then use the app to scan the QR code. For example, you might choose the camera icon or choose an option similar to **Scan code**, and then use the device's camera to scan the code.
- In the **Set up device** wizard, choose **Show secret key**, and then type the secret key into your MFA app.

When you are finished, the virtual MFA device starts generating one-time passwords.

8. In the **Set up device** wizard, in the **Enter the code from your authenticator app.** box, type the one-time password that currently appears in the virtual MFA device. Choose **Register MFA**.

> **Important**
>
> Submit your request immediately after generating the code. If you generate the codes and then wait too long to submit the request, the MFA device is successfully associated with the user. However, the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resync the device.

The virtual MFA device is now ready to use with AWS.

9. Sign out of the console and then sign in as **`MFAUser`** again. This time AWS prompts you for an MFA code from your phone. When you get it, type the code in the box and then choose **Submit**.
10. Choose **EC2** to open the Amazon EC2 console again. Note that this time you can see all the information and perform any actions you want. If you go to any other console as this user, you see access denied messages. The reason is that the policies in this tutorial grant access only to Amazon EC2.

Related resources

For additional information, see the following topics:

- AWS Multi-factor authentication in IAM
- MFA enabled sign-in

IAM tutorial: Use a CloudFormation template to create a SAML Identity Provider (IdP)

To set up SAML federation for your AWS account, you need to create a SAML Identity Provider (IdP). This tutorial shows you how to use a CloudFormation template to create a SAML IdP that establishes trust between AWS and your external IdP.

The template creates a SAML IdP configured with your IdP's metadata document. Federated IAM roles can then reference this IdP to allow authenticated users from your external IdP to access AWS resources.

The deployed resource consists of a SAML IdP configured with your IdP's metadata document and optional encryption settings.

Prerequisites

This tutorial assumes that you have the following already in place:

- Python 3.6 or later installed on your local machine to run the Python command used in this tutorial for formatting your IdP's SAML metadata XML file.
- A SAML metadata document from your external IdP saved as an XML file.

Create a SAML IdP using CloudFormation

To create the SAML IdP, create a CloudFormation template and use it to create a stack containing the IdP resource.

Create the template

First, create the CloudFormation template.

1. In the Template section, choose the copy icon on the **JSON** or **YAML** tab to copy the template contents.
2. Paste the template contents into a new file.
3. Save the file locally.

Create the stack

Next, use the template you've saved to provision a CloudFormation stack.

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. On the **Stacks** page, from the **Create stack** menu, choose **with new resources (standard)**.
3. Specify the template:
   a. Under **Prerequisite**, choose **Choose an existing template**.
   b. Under **Specify template**, choose **Upload a template file**.
   c. Choose **Choose file**, navigate to the template file, and choose it.
   d. Choose **Next**.
4. Specify the following stack details:
   a. Enter a stack name.
   b. For **IdentityProviderName**, you can leave this empty to auto-generate a name based on the stack name, or enter a custom name for your SAML IdP. Custom names must contain only alphanumeric characters, periods, underscores, and hyphens.
   c. For **IdentityProviderSAMLMetadataDocument**, you need to format your SAML metadata XML file as a single line before pasting it into this field. This is necessary because the CloudFormation console requires XML content to be formatted as a single line when passed through console parameters.

      Use the following Python command to reformat your XML file:

      ```
      python3 -c "import sys, re; content=open(sys.argv[1]).read(); print(re.sub(r'>\s+<', '><', content.replace('\n', '').replace('\r', '').strip()))" saml-metadata.xml
      ```

      > **Note**
      >
      > The IdP's SAML metadata document must be formatted as a single line for console parameter input. The Python command removes line breaks and extra whitespace to create the required format while maintaining all original content and structure.

Copy the output from the Python command and paste it into the **IdentityProviderSAMLMetadataDocument** field.

Example of formatted SAML metadata document (abbreviated):

```
<?xml version="1.0" encoding="UTF-8"?><md:EntityDescriptor
 xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
 entityID="https://portal.sso.example.com/saml/assertion/
CompanyIdP"><md:IDPSSODescriptor WantAuthnRequestsSigned="false"
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol"><md:KeyDescriptor
 use="signing"><ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/
xmldsig#"><ds:X509Data><ds:X509Certificate>MIIDXTCCAkWgAwIBAgIJAJC1HiIAZAiIMA0GCSqGSIb3
ds:X509Certificate></ds:X509Data></ds:KeyInfo></
md:KeyDescriptor><md:SingleLogoutService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
 Location="https://portal.sso.example.com/saml/logout/
CompanyIdP"/><md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:persistent</md:NameIDFormat><md:SingleSignOnService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://
portal.sso.example.com/saml/assertion/CompanyIdP"/></md:IDPSSODescriptor></
md:EntityDescriptor>
```

d. For other parameters, accept the default values or enter your own based on your requirements:

- **IdentityProviderAddPrivateKey** - Optional private key for decrypting SAML assertions
- **IdentityProviderAssertionEncryptionMode** - Optional, sets encryption mode for SAML assertions (Allowed, Required, or empty)

e. Choose **Next**.

5. Configure the stack options:

a. Under **Stack failure options**, choose **Delete all newly created resources**.

> **Note**
>
> Choosing this option prevents you from possibly being billed for resources whose deletion policy specifies they be retained even if the stack creation fails.

b. Accept all other default values.

c. Under **Capabilities**, check the box to acknowledge that CloudFormation might create IAM resources in your account.

d. Choose **Next**.

6. Review the stack details and choose **Submit**.

CloudFormation creates the stack. After the stack creation is complete, the stack resources are ready to use. You can use the **Resources** tab on the stack detail page to view the resources that were provisioned in your account.

The stack will output the following values, which you can view on the **Outputs** tab:

- **ProviderARN**: The ARN of the created SAML IdP (for example, `arn:aws:iam::123456789012:saml-provider/CompanyIdP`). You'll need this ARN when creating roles that trust this provider.
- **ProviderName**: The name of the created SAML IdP (for example, `CompanyIdP` if you specified a custom name, or `my-saml-stack-saml-provider` if you used the default naming).

These outputs are also exported, allowing them to be imported by other CloudFormation stacks using the `Fn::ImportValue` function.

Verify the SAML IdP

After the SAML IdP has been created, you can verify its configuration and note its ARN for use with federated roles.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers**.

   You should see your newly created SAML IdP in the list.
3. Choose the IdP name to view its details.

   On the IdP detail page, you can see the SAML metadata document and other configuration details.
4. Note the **Provider ARN** displayed on the details page.

   You will need this ARN when creating federated IAM roles that trust this IdP.

5. Review the metadata document to make sure it matches what you provided from your external IdP.

Your SAML IdP is now ready to be used by federated IAM roles. You can create roles that trust this IdP to allow authenticated users from your external IdP to assume those roles and access AWS resources.

Clean up: delete resources

As a final step, you'll delete the stack and the resources it contains.

1. Open the CloudFormation console.
2. On the **Stacks** page, choose the stack created from the template, and choose **Delete**, then confirm **Delete**.

   CloudFormation initiates deletion of the stack and all resources it includes.

CloudFormation template details

Resources

The CloudFormation template for this tutorial will create the following resource in your account:

`AWS::IAM::SAMLProvider`: A SAML IdP that establishes trust between AWS and your external IdP.

Configuration

The template includes the following configurable parameters:

- **IdentityProviderName** - The name for your SAML IdP (leave empty for auto-generated name)

  Example: `CompanyIdP` or `EnterpriseSSO`
- **IdentityProviderSAMLMetadataDocument** - The SAML metadata document from your external IdP (formatted as a single line)
- **IdentityProviderAddPrivateKey** - Optional private key for decrypting SAML assertions
- **IdentityProviderAssertionEncryptionMode** - Optional, sets encryption mode for SAML assertions

CloudFormation template

Save the following JSON or YAML code as a separate file to use as the CloudFormation template for this tutorial.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "[AWSDocs] IAM: tutorial_saml-idp",
  "Parameters": {
    "IdentityProviderName": {
      "Type": "String",
      "Description": "Name of the SAML Identity Provider (leave empty for auto-
generated name like '{StackName}-{UniqueId}')",
      "Default": "",
      "AllowedPattern": "^$|^[a-zA-Z0-9._-]+$",
      "ConstraintDescription": "Must be empty or contain only alphanumeric
 characters, periods, underscores, and hyphens"
    },
    "IdentityProviderSAMLMetadataDocument": {
      "Type": "String",
      "Description": "SAML metadata document from identity provider"
    },
    "IdentityProviderAddPrivateKey": {
      "Type": "String",
      "Description": "Optional private key for decrypting SAML assertions. The
 private key must be a .pem file that uses AES-GCM or AES-CBC encryption algorithm
 to decrypt SAML assertions.",
      "Default": ""
    },
    "IdentityProviderAssertionEncryptionMode": {
      "Type": "String",
      "Description": "Optional, sets encryption mode for SAML assertions",
      "Default": "",
      "AllowedValues": ["", "Allowed", "Required"]
    }
  },
  "Conditions": {
    "HasPrivateKey": {"Fn::Not": [{"Fn::Equals": [{"Ref":
 "IdentityProviderAddPrivateKey"}, ""]}]},
    "HasEncryptionMode": {"Fn::Not": [{"Fn::Equals": [{"Ref":
 "IdentityProviderAssertionEncryptionMode"}, ""]}]},
```

```
    "HasCustomName": {"Fn::Not": [{"Fn::Equals": [{"Ref": "IdentityProviderName"},
 ""]}]}
  },
  "Resources": {
    "SAMLProvider": {
      "Type": "AWS::IAM::SAMLProvider",
      "Properties": {
        "Name": {"Fn::If": ["HasCustomName", {"Ref": "IdentityProviderName"},
 {"Ref": "AWS::NoValue"}]},
        "SamlMetadataDocument": {"Ref": "IdentityProviderSAMLMetadataDocument"},
        "Tags": [
          {
            "Key": "Name",
            "Value": {"Fn::If": ["HasCustomName", {"Ref": "IdentityProviderName"},
 {"Fn::Sub": "${AWS::StackName}-saml-provider"}]}
          }
        ],
        "AddPrivateKey": {"Fn::If": ["HasPrivateKey", {"Ref":
 "IdentityProviderAddPrivateKey"}, {"Ref": "AWS::NoValue"}]},
        "AssertionEncryptionMode": {"Fn::If": ["HasEncryptionMode", {"Ref":
 "IdentityProviderAssertionEncryptionMode"}, {"Ref": "AWS::NoValue"}]}
      }
    }
  },
  "Outputs": {
    "ProviderARN": {
      "Description": "ARN of the created SAML Identity Provider",
      "Value": {"Ref": "SAMLProvider"},
      "Export": {
        "Name": {"Fn::Sub": "${AWS::StackName}-ProviderARN"}
      }
    },
    "ProviderName": {
      "Description": "Name of the SAML Identity Provider",
      "Value": {"Fn::If": ["HasCustomName", {"Ref": "IdentityProviderName"},
 {"Fn::Sub": "${AWS::StackName}-saml-provider"}]},
      "Export": {
        "Name": {"Fn::Sub": "${AWS::StackName}-ProviderName"}
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] IAM: tutorial_saml-idp'

Parameters:
  IdentityProviderName:
    Type: String
    Description: Name of the SAML Identity Provider (leave empty for auto-generated
 name like '{StackName}-{UniqueId}')
    Default: ""
    AllowedPattern: '^$|^[a-zA-Z0-9._-]+$'
    ConstraintDescription: 'Must be empty or contain only alphanumeric characters,
 periods, underscores, and hyphens'

  IdentityProviderSAMLMetadataDocument:
    Type: String
    Description: SAML metadata document from identity provider

  IdentityProviderAddPrivateKey:
    Type: String
    Description: Optional private key for decrypting SAML assertions. The private
 key must be a .pem file that uses AES-GCM or AES-CBC encryption algorithm to
 decrypt SAML assertions.
    Default: ""

  IdentityProviderAssertionEncryptionMode:
    Type: String
    Description: Optional, sets encryption mode for SAML assertions
    Default: ""
    AllowedValues:
      - ""
      - "Allowed"
      - "Required"

Conditions:
  HasPrivateKey: !Not [!Equals [!Ref IdentityProviderAddPrivateKey, ""]]
  HasEncryptionMode: !Not [!Equals [!Ref IdentityProviderAssertionEncryptionMode,
 ""]]
  HasCustomName: !Not [!Equals [!Ref IdentityProviderName, ""]]

Resources:
  SAMLProvider:
    Type: 'AWS::IAM::SAMLProvider'
```

```
    Properties:
      Name: !If
        - HasCustomName
        - !Ref IdentityProviderName
        - !Ref AWS::NoValue
      SamlMetadataDocument: !Ref IdentityProviderSAMLMetadataDocument
      Tags:
        - Key: Name
          Value: !If
            - HasCustomName
            - !Ref IdentityProviderName
            - !Sub '${AWS::StackName}-saml-provider'
      AddPrivateKey: !If
        - HasPrivateKey
        - !Ref IdentityProviderAddPrivateKey
        - !Ref AWS::NoValue
      AssertionEncryptionMode: !If
        - HasEncryptionMode
        - !Ref IdentityProviderAssertionEncryptionMode
        - !Ref AWS::NoValue

Outputs:
  ProviderARN:
    Description: 'ARN of the created SAML Identity Provider'
    Value: !Ref SAMLProvider
    Export:
      Name: !Sub '${AWS::StackName}-ProviderARN'

  ProviderName:
    Description: 'Name of the SAML Identity Provider'
    Value: !If
      - HasCustomName
      - !Ref IdentityProviderName
      - !Sub '${AWS::StackName}-saml-provider'
    Export:
      Name: !Sub '${AWS::StackName}-ProviderName'
```

IAM tutorial: Use a CloudFormation template to create a SAML federated IAM role

When you have an existing SAML Identity Provider (IdP) configured in your AWS account, you can create federated IAM roles that trust that IdP. This tutorial shows you how to use a CloudFormation template to create a SAML federated IAM role that can be assumed by users authenticated through your external IdP.

The template creates a federated IAM role with a trust policy that allows your SAML IdP to assume the role. Users authenticated by your external IdP can assume this role to access AWS resources based on the permissions attached to the role.

The deployed resource consists of the following:

- A federated IAM role that trusts your existing SAML IdP.
- Configurable managed policies that can be attached to the role to grant specific permissions.
- Optional permissions boundary and session duration settings.

Prerequisites

This tutorial assumes that you have the following already in place:

- An existing SAML IdP configured in your AWS account. If you don't have one, you can create it using the IAM tutorial: Use a CloudFormation template to create a SAML Identity Provider (IdP) tutorial.
- The ARN of your SAML IdP, which you'll need to specify as a parameter when creating the stack.
- Python 3.6 or later installed on your local machine to run the Python command used in this tutorial for formatting your IdP's SAML metadata XML file.

Create a SAML federated role using CloudFormation

To create the SAML federated role, you'll create a CloudFormation template and use it to create a stack containing the role.

Create the template

First, create the CloudFormation template.

1. In the Template section, click the copy icon on the **JSON** or **YAML** tab to copy the template contents.
2. Paste the template contents into a new file.
3. Save the file locally.

Create the stack

Next, use the template you've saved to provision a CloudFormation stack.

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. On the **Stacks** page, from the **Create stack** menu, choose **with new resources (standard)**.
3. Specify the template:
   a. Under **Prerequisite**, choose **Choose an existing template**.
   b. Under **Specify template**, choose **Upload a template file**.
   c. Choose **Choose file**, navigate to the template file, and choose it.
   d. Choose **Next**.
4. Specify the following stack details:
   a. Enter a stack name.
   b. For **SAMLProviderARN**, enter the ARN of your existing SAML IdP. This should be in the format `arn:aws:iam::123456789012:saml-provider/YourProviderName`.

      Example: `arn:aws:iam::123456789012:saml-provider/CompanyIdP`

      > **Note**
      >
      > If you created your SAML IdP using the IAM tutorial: Use a CloudFormation template to create a SAML Identity Provider (IdP) tutorial, you can find the provider ARN in the Outputs tab of that CloudFormation stack.

   c. For **RoleName**, you can leave this empty to auto-generate a name based on the stack name, or enter a custom name for the IAM role.

      Example: `SAML-Developer-Access` or `SAML-ReadOnly-Role`
   d. For other parameters, accept the default values or enter your own based on your requirements:

- **RoleSessionDuration** - Maximum session duration in seconds (3600-43200, default 7200)

  Example: `14400` (4 hours)
- **RolePermissionsBoundary** - Optional ARN of a permissions boundary policy

  Example: `arn:aws:iam::123456789012:policy/DeveloperBoundary`
- **RolePath** - Path for the IAM role (default is /)

  Example: `/saml-roles/`
- **ManagedPolicy1-5** - Optional ARNs of up to 5 managed policies to attach

  Example for ManagedPolicy1: `arn:aws:iam::aws:policy/ReadOnlyAccess`

  Example for ManagedPolicy2: `arn:aws:iam::123456789012:policy/CustomPolicy`

e. Choose **Next**.

5. Configure the stack options:

   a. Under **Stack failure options**, choose **Delete all newly created resources**.

   > **Note**
   >
   > Choosing this option prevents you from possibly being billed for resources whose deletion policy specifies they be retained even if the stack creation fails.

   b. Accept all other default values.

   c. Under **Capabilities**, check the box to acknowledge that CloudFormation might create IAM resources in your account.

   d. Choose **Next**.

6. Review the stack details and choose **Submit**.

CloudFormation creates the stack. Once the stack creation is complete, the stack resources are ready to use. You can use the **Resources** tab on the stack detail page to view the resources that were provisioned in your account.

The stack will output the following value, which you can view on the **Outputs** tab:

- **RoleARN**: The ARN of the created IAM role (for example, `arn:aws:iam::123456789012:role/SAML-Developer-Access` or `arn:aws:iam::123456789012:role/stack-name-a1b2c3d4` if using auto-generated name).

You'll need this role ARN when configuring your IdP to send the appropriate SAML attributes for role assumption.

Test the SAML federated role

Once the SAML federated role has been created, you can verify its configuration and test the federation setup.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Find and choose your newly created federated role.

   If you provided a custom role name, look for that name. If you left the RoleName parameter empty, the role will have an auto-generated name based on the stack name and a unique identifier.
4. Choose the **Trust relationships** tab to review the trust policy.

   The trust policy should show that your SAML IdP is trusted to assume this role with the condition that the SAML audience (`SAML:aud`) matches `https://signin.aws.amazon.com/saml`.
5. Choose the **Permissions** tab to review the attached policies.

   You can see any managed policies that were attached to the role during creation.
6. Note the **Role ARN** displayed on the role summary page.

   You will need this ARN to configure your external IdP to allow users to assume this role.

Your SAML federated role is now ready to be used. Configure your external IdP to include this role's ARN in SAML assertions, and authenticated users will be able to assume this role to access AWS resources.

Clean up: delete resources

As a final step, you'll delete the stack and the resources it contains.

1. Open the CloudFormation console.
2. On the **Stacks** page, choose the stack created from the template, and choose **Delete**, then confirm **Delete**.

   CloudFormation initiates deletion of the stack and all resources it includes.

CloudFormation template details

Resources

The CloudFormation template for this tutorial will create the following resource in your account:

- `AWS::IAM::Role`: A federated IAM role that can be assumed by users authenticated through your SAML IdP.

Configuration

The template includes the following configurable parameters:

- **RoleName** - Name of the IAM Role (leave empty for auto-generated name)
- **SAMLProviderARN** - ARN of the SAML IdP (required)
- **RoleSessionDuration** - Maximum session duration in seconds (3600-43200, default 7200)
- **RolePermissionsBoundary** - Optional ARN of permissions boundary policy
- **RolePath** - Path for the IAM role (default /)
- **ManagedPolicy1-5** - Optional ARNs of up to 5 managed policies to attach

CloudFormation template

Save the following JSON or YAML code as a separate file to use as the CloudFormation template for this tutorial.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "[AWSDocs] IAM: tutorial_saml-federated-role",
  "Parameters": {
    "RoleName": {
      "Type": "String",
      "Description": "Name of the IAM Role (leave empty for auto-generated name like
 '{StackName}-{UniqueId}')",
      "Default": "",
      "AllowedPattern": "^$|^[\\w+=,.@-]{1,64}$",
      "ConstraintDescription": "Must be empty or 1-64 characters and can contain
 alphanumeric characters and +=,.@-"
    },
    "SAMLProviderARN": {
      "Type": "String",
      "Description": "ARN of the SAML Identity Provider",
      "AllowedPattern": "^arn:aws:iam::\\d{12}:saml-provider/[a-zA-Z0-9._-]+$",
      "ConstraintDescription": "Must be a valid SAML provider ARN"
    },
    "RoleSessionDuration": {
      "Type": "Number",
      "Description": "The maximum session duration (in seconds) that you want to set
 for the specified role (3600-43200)",
      "MinValue": 3600,
      "MaxValue": 43200,
      "Default": 7200
    },
    "RolePermissionsBoundary": {
      "Type": "String",
      "Description": "Optional ARN of the permissions boundary policy (leave empty
 for none)",
      "Default": ""
    },
    "RolePath": {
      "Type": "String",
      "Description": "Path for the IAM role (must start and end with /)",
      "Default": "/",
      "AllowedPattern": "^\/.*\/$|^\/$",
      "ConstraintDescription": "Role path must start and end with forward slash (/)"
    },
    "RoleManagedPolicy1": {
      "Type": "String",
```

```
      "Description": "Optional managed policy ARN 1",
      "Default": ""
    },
    "RoleManagedPolicy2": {
      "Type": "String",
      "Description": "Optional managed policy ARN 2",
      "Default": ""
    },
    "RoleManagedPolicy3": {
      "Type": "String",
      "Description": "Optional managed policy ARN 3",
      "Default": ""
    },
    "RoleManagedPolicy4": {
      "Type": "String",
      "Description": "Optional managed policy ARN 4",
      "Default": ""
    },
    "RoleManagedPolicy5": {
      "Type": "String",
      "Description": "Optional managed policy ARN 5",
      "Default": ""
    }
  },
  "Conditions": {
    "HasCustomRoleName": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleName"}, ""]}]},
    "HasPermissionsBoundary": {"Fn::Not": [{"Fn::Equals": [{"Ref":
"RolePermissionsBoundary"}, ""]}]},
    "HasPolicy1": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy1"},
""]}]},
    "HasPolicy2": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy2"},
""]}]},
    "HasPolicy3": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy3"},
""]}]},
    "HasPolicy4": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy4"},
""]}]},
    "HasPolicy5": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy5"}, ""]}]}
  },
  "Resources": {
    "SAMLFederatedRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "RoleName": {"Fn::If": ["HasCustomRoleName", {"Ref": "RoleName"}, {"Ref":
"AWS::NoValue"}]},
```

```
        "Description": "IAM role with SAML provider trust",
        "MaxSessionDuration": {"Ref": "RoleSessionDuration"},
        "PermissionsBoundary": {"Fn::If": ["HasPermissionsBoundary", {"Ref":
"RolePermissionsBoundary"}, {"Ref": "AWS::NoValue"}]},
        "Path": {"Ref": "RolePath"},
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Federated": {"Ref": "SAMLProviderARN"}
              },
              "Action": "sts:AssumeRoleWithSAML",
              "Condition": {
                "StringEquals": {
                  "SAML:aud": "https://signin.aws.amazon.com/saml"
                }
              }
            }
          ]
        },
        "ManagedPolicyArns": {
          "Fn::Split": [
            ",",
            {
              "Fn::Join": [
                ",",
                [
                  {"Fn::If": ["HasPolicy1", {"Ref": "RoleManagedPolicy1"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy2", {"Ref": "RoleManagedPolicy2"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy3", {"Ref": "RoleManagedPolicy3"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy4", {"Ref": "RoleManagedPolicy4"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy5", {"Ref": "RoleManagedPolicy5"}, {"Ref":
"AWS::NoValue"}]}
                ]
              ]
            }
          ]
        }
```

```
      }
    }
  },
  "Outputs": {
    "RoleARN": {
      "Description": "ARN of the created IAM Role",
      "Value": {"Fn::GetAtt": ["SAMLFederatedRole", "Arn"]},
      "Export": {
        "Name": {"Fn::Sub": "${AWS::StackName}-RoleARN"}
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] IAM: tutorial_saml-federated-role'

Parameters:
  RoleName:
    Type: String
    Description: 'Name of the IAM Role (leave empty for auto-generated name like
 ''{StackName}-{UniqueId}'')'
    Default: ""
    AllowedPattern: '^$|^[\w+=,.@-]{1,64}$'
    ConstraintDescription: 'Must be empty or 1-64 characters and can contain
 alphanumeric characters and +=,.@-'

  SAMLProviderARN:
    Type: String
    Description: 'ARN of the SAML Identity Provider'
    AllowedPattern: '^arn:aws:iam::\d{12}:saml-provider/[a-zA-Z0-9._-]+$'
    ConstraintDescription: 'Must be a valid SAML provider ARN'

  RoleSessionDuration:
    Type: Number
    Description: 'The maximum session duration (in seconds) that you want to set for
 the specified role (3600-43200)'
    MinValue: 3600
    MaxValue: 43200
    Default: 7200
```

```
  RolePermissionsBoundary:
    Type: String
    Description: Optional ARN of the permissions boundary policy (leave empty for
 none)
    Default: ""

  RolePath:
    Type: String
    Description: 'Path for the IAM role (must start and end with /)'
    Default: "/"
    AllowedPattern: '^\/.*\/$|^\/$'
    ConstraintDescription: 'Role path must start and end with forward slash (/)'

  RoleManagedPolicy1:
    Type: String
    Description: Optional managed policy ARN 1
    Default: ""
  RoleManagedPolicy2:
    Type: String
    Description: Optional managed policy ARN 2
    Default: ""
  RoleManagedPolicy3:
    Type: String
    Description: Optional managed policy ARN 3
    Default: ""
  RoleManagedPolicy4:
    Type: String
    Description: Optional managed policy ARN 4
    Default: ""
  RoleManagedPolicy5:
    Type: String
    Description: Optional managed policy ARN 5
    Default: ""

Conditions:
  HasCustomRoleName: !Not [!Equals [!Ref RoleName, ""]]
  HasPermissionsBoundary: !Not [!Equals [!Ref RolePermissionsBoundary, ""]]
  HasPolicy1: !Not [!Equals [!Ref RoleManagedPolicy1, ""]]
  HasPolicy2: !Not [!Equals [!Ref RoleManagedPolicy2, ""]]
  HasPolicy3: !Not [!Equals [!Ref RoleManagedPolicy3, ""]]
  HasPolicy4: !Not [!Equals [!Ref RoleManagedPolicy4, ""]]
  HasPolicy5: !Not [!Equals [!Ref RoleManagedPolicy5, ""]]

Resources:
```

```
  SAMLFederatedRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: !If
        - HasCustomRoleName
        - !Ref RoleName
        - !Ref AWS::NoValue
      Description: 'IAM role with SAML provider trust'
      MaxSessionDuration: !Ref RoleSessionDuration
      PermissionsBoundary: !If
        - HasPermissionsBoundary
        - !Ref RolePermissionsBoundary
        - !Ref AWS::NoValue
      Path: !Ref RolePath
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Federated: !Ref SAMLProviderARN
            Action: 'sts:AssumeRoleWithSAML'
            Condition:
              StringEquals:
                'SAML:aud': 'https://signin.aws.amazon.com/saml'
      ManagedPolicyArns:
        !Split
          - ','
          - !Join
            - ','
            - - !If [HasPolicy1, !Ref RoleManagedPolicy1, !Ref 'AWS::NoValue']
              - !If [HasPolicy2, !Ref RoleManagedPolicy2, !Ref 'AWS::NoValue']
              - !If [HasPolicy3, !Ref RoleManagedPolicy3, !Ref 'AWS::NoValue']
              - !If [HasPolicy4, !Ref RoleManagedPolicy4, !Ref 'AWS::NoValue']
              - !If [HasPolicy5, !Ref RoleManagedPolicy5, !Ref 'AWS::NoValue']

Outputs:
  RoleARN:
    Description: 'ARN of the created IAM Role'
    Value: !GetAtt SAMLFederatedRole.Arn
    Export:
      Name: !Sub '${AWS::StackName}-RoleARN'
```

IAM tutorial: Use a CloudFormation template to create a SAML Identity Provider (IdP) and SAML federated IAM role

To get familiar with SAML federation and its capabilities, you'll use a CloudFormation template to set up a SAML Identity Provider (IdP) and associated federated IAM role. This tutorial shows you how to create both resources together in a single stack.

The template creates a SAML IdP that can be used for federated access to AWS resources, along with an IAM role that trusts the SAML provider. Users authenticated by your external IdP can assume this role to access AWS resources.

The deployed resources consist of the following:

- A SAML IdP configured with your IdP's metadata document.
- A federated IAM role that trusts the SAML IdP and can be assumed by authenticated users.
- Configurable managed policies that can be attached to the role to grant specific permissions.

Prerequisites

This tutorial assumes that you have the following already in place:

- Python 3.6 or later installed on your local machine to run the Python command used in this tutorial for formatting your IdP's SAML metadata XML file.
- A SAML metadata document from your external IdP saved as an XML file.

Create a SAML IdP and role using CloudFormation

To create the SAML IdP and federated role, you'll create a CloudFormation template and use it to create a stack containing both resources.

Create the template

First, create the CloudFormation template.

1. In the Template section, click the copy icon on the **JSON** or **YAML** tab to copy the template contents.

2. Paste the template contents into a new file.
3. Save the file locally.

Create the stack

Next, use the template you've saved to provision a CloudFormation stack.

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. On the **Stacks** page, from the **Create stack** menu, choose **with new resources (standard)**.
3. Specify the template:
   a. Under **Prerequisite**, choose **Choose an existing template**.
   b. Under **Specify template**, choose **Upload a template file**.
   c. Choose **Choose file**, navigate to the template file, and choose it.
   d. Choose **Next**.
4. Specify the following stack details:
   a. Enter a stack name.
   b. For **IdentityProviderName**, you can leave this empty to auto-generate a name based on the stack name, or enter a custom name for your SAML IdP.

      Example: `CompanyIdP` or `EnterpriseSSO`
   c. For **IdentityProviderSAMLMetadataDocument**, you need to format your SAML metadata XML file as a single line before pasting it into this field. This is necessary because the CloudFormation console requires XML content to be formatted as a single line when passed through console parameters.

      Use the following Python command to reformat your XML file:

```
python3 -c "import sys, re; content=open(sys.argv[1]).read(); print(re.sub(r'>
\s+<', '><', content.replace('\n', '').replace('\r', '').strip()))" saml-
metadata.xml
```

> **Note**
>
> The IdP's SAML metadata document must be formatted as a single line for console parameter input. The Python command removes line breaks and extra whitespace to create the required format while maintaining all original content and structure.

Copy the output from the Python command and paste it into the **IdentityProviderSAMLMetadataDocument** field.

Example of formatted SAML metadata document (abbreviated):

```
<?xml version="1.0" encoding="UTF-8"?><md:EntityDescriptor
 xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
 entityID="https://portal.sso.example.com/saml/assertion/
CompanyIdP"><md:IDPSSODescriptor WantAuthnRequestsSigned="false"
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol"><md:KeyDescriptor
 use="signing"><ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/
xmldsig#"><ds:X509Data><ds:X509Certificate>MIIDXTCCAkWgAwIBAgIJAJC1HiIAZAiIMA0GCSqGSIb3
ds:X509Certificate></ds:X509Data></ds:KeyInfo></
md:KeyDescriptor><md:SingleLogoutService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
 Location="https://portal.sso.example.com/saml/logout/
CompanyIdP"/><md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:persistent</md:NameIDFormat><md:SingleSignOnService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://
portal.sso.example.com/saml/assertion/CompanyIdP"/></md:IDPSSODescriptor></
md:EntityDescriptor>
```

d. For **RoleName**, you can leave this empty to auto-generate a name based on the stack name, or enter a custom name for the federated IAM role.

   Example: `SAML-Developer-Access` or `SAML-ReadOnly-Role`

e. For other parameters, accept the default values or enter your own based on your requirements:

   - **IdentityProviderAddPrivateKey** - Optional private key for decrypting SAML assertions
   - **IdentityProviderAssertionEncryptionMode** - Encryption mode for SAML assertions

     Example values: `Allowed`, `Required`, or leave empty for no encryption

- **RoleSessionDuration** - Maximum session duration in seconds (3600-43200, default 7200)

  Example: `14400` (4 hours)

- **RolePermissionsBoundary** - Optional ARN of a permissions boundary policy

  Example: `arn:aws:iam::123456789012:policy/DeveloperBoundary`

- **RolePath** - Path for the IAM role (default is /)

  Example: `/saml-roles/`

- **RoleManagedPolicy1-5** - Optional ARNs of up to 5 managed policies to attach

  Example for RoleManagedPolicy1: `arn:aws:iam::aws:policy/ReadOnlyAccess`

  Example for RoleManagedPolicy2: `arn:aws:iam::123456789012:policy/CustomPolicy`

f. Choose **Next**.

5. Configure the stack options:

   a. Under **Stack failure options**, choose **Delete all newly created resources**.

   > **Note**
   >
   > Choosing this option prevents you from possibly being billed for resources whose deletion policy specifies they be retained even if the stack creation fails.

   b. Accept all other default values.

   c. Under **Capabilities**, check the box to acknowledge that CloudFormation might create IAM resources in your account.

   d. Choose **Next**.

6. Review the stack details and choose **Submit**.

CloudFormation creates the stack. Once the stack creation is complete, the stack resources are ready to use. You can use the **Resources** tab on the stack detail page to view the resources that were provisioned in your account.

The stack will output the following values, which you can view on the **Outputs** tab:

- **RoleARN**: The ARN of the created IAM role (for example, `arn:aws:iam::123456789012:role/SAML-Developer-Access` or `arn:aws:iam::123456789012:role/stack-name-a1b2c3d4` if using auto-generated name).
- **IdentityProviderARN**: The ARN of the created SAML IdP (for example, `arn:aws:iam::123456789012:saml-provider/CompanyIdP`).

You'll need both of these ARNs when configuring your IdP to send the appropriate SAML attributes for role assumption.

Test the SAML federation

Once the SAML IdP and federated role have been created, you can test the federation setup.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers**.

   You should see your newly created SAML IdP in the list.
3. Choose the IdP name to view its details.

   On the IdP detail page, you can see the SAML metadata document and other configuration details.
4. In the navigation pane, choose **Roles**.
5. Find and choose your newly created federated role.

   On the role detail page, you can see the trust policy that allows the SAML IdP to assume this role.
6. Choose the **Trust relationships** tab to review the trust policy.

   The trust policy should show that the SAML IdP is trusted to assume this role with the condition that the SAML audience (`SAML:aud`) matches `https://signin.aws.amazon.com/saml`.

Clean up: delete resources

As a final step, you'll delete the stack and the resources it contains.

1. Open the CloudFormation console.
2. On the **Stacks** page, choose the stack created from the template, and choose **Delete**, then confirm **Delete**.

   CloudFormation initiates deletion of the stack and all resources it includes.

CloudFormation template details

Resources

The CloudFormation template for this tutorial will create the following resources in your account:

- `AWS::IAM::SAMLProvider`: A SAML IdP that establishes trust between AWS and your external IdP.
- `AWS::IAM::Role`: A federated IAM role that can be assumed by users authenticated through the SAML IdP.

Configuration

The template includes the following configurable parameters:

- **IdentityProviderName** - Name of the SAML IdP (leave empty for auto-generated name)
- **IdentityProviderSAMLMetadataDocument** - SAML metadata document from your IdP (required)
- **IdentityProviderAddPrivateKey** - Optional private key for decrypting SAML assertions
- **IdentityProviderAssertionEncryptionMode** - Encryption mode for SAML assertions
- **RoleName** - Name of the IAM Role (leave empty for auto-generated name)
- **RolePath** - Path for the IAM role (default /)
- **RolePermissionsBoundary** - Optional ARN of permissions boundary policy
- **RoleSessionDuration** - Maximum session duration in seconds (3600-43200, default 7200)
- **RoleManagedPolicy1-5** - Optional ARNs of up to 5 managed policies to attach

CloudFormation template

Save the following JSON or YAML code as a separate file to use as the CloudFormation template for this tutorial.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "[AWSDocs] IAM: tutorial_saml-idp-and-federated-role",
  "Parameters": {
    "IdentityProviderName": {
      "Type": "String",
      "Description": "Name of the SAML Identity Provider (leave empty for auto-
generated name like '{StackName}-{UniqueId}')",
      "Default": "",
      "AllowedPattern": "^$|^[a-zA-Z0-9._-]+$",
      "ConstraintDescription": "Must be empty or contain only alphanumeric
 characters, periods, underscores, and hyphens"
    },
    "IdentityProviderSAMLMetadataDocument": {
      "Type": "String",
      "Description": "SAML metadata document from identity provider"
    },
    "IdentityProviderAddPrivateKey": {
      "Type": "String",
      "Description": "Optional private key for decrypting SAML assertions. The
 private key must be a .pem file that uses AES-GCM or AES-CBC encryption algorithm
 to decrypt SAML assertions.",
      "Default": ""
    },
    "IdentityProviderAssertionEncryptionMode": {
      "Type": "String",
      "Description": "Optional, sets encryption mode for SAML assertions",
      "Default": "",
      "AllowedValues": ["", "Allowed", "Required"]
    },
    "RoleName": {
      "Type": "String",
      "Description": "Name of the IAM Role (leave empty for auto-generated name like
 '{StackName}-{UniqueId}')",
      "Default": "",
      "AllowedPattern": "^$|^[\\w+=,.@-]{1,64}$",
      "ConstraintDescription": "Must be empty or 1-64 characters and can contain
 alphanumeric characters and +=,.@-"
    },
    "RolePath": {
      "Type": "String",
      "Description": "Path for the IAM Role",
```

```
      "AllowedPattern": "(^\\/$)|(^\\/.*\\/$)",
      "Default": "/"
    },
    "RolePermissionsBoundary": {
      "Type": "String",
      "Description": "Optional ARN of the permissions boundary policy (leave empty
for none)",
      "Default": ""
    },
    "RoleSessionDuration": {
      "Description": "The maximum session duration (in seconds) that you want to set
for the specified role (3600-43200)",
      "Type": "Number",
      "MinValue": 3600,
      "MaxValue": 43200,
      "Default": 7200
    },
    "RoleManagedPolicy1": {
      "Type": "String",
      "Description": "Optional managed policy ARN 1",
      "Default": ""
    },
    "RoleManagedPolicy2": {
      "Type": "String",
      "Description": "Optional managed policy ARN 2",
      "Default": ""
    },
    "RoleManagedPolicy3": {
      "Type": "String",
      "Description": "Optional managed policy ARN 3",
      "Default": ""
    },
    "RoleManagedPolicy4": {
      "Type": "String",
      "Description": "Optional managed policy ARN 4",
      "Default": ""
    },
    "RoleManagedPolicy5": {
      "Type": "String",
      "Description": "Optional managed policy ARN 5",
      "Default": ""
    }
  },
  "Conditions": {
```

```
   "HasCustomProviderName": {"Fn::Not": [{"Fn::Equals": [{"Ref":
"IdentityProviderName"}, ""]}]},
   "HasCustomRoleName": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleName"}, ""]}]},
   "HasPermissionsBoundary": {"Fn::Not": [{"Fn::Equals": [{"Ref":
"RolePermissionsBoundary"}, ""]}]},
   "HasPolicy1": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy1"},
""]}]},
   "HasPolicy2": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy2"},
""]}]},
   "HasPolicy3": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy3"},
""]}]},
   "HasPolicy4": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy4"},
""]}]},
   "HasPolicy5": {"Fn::Not": [{"Fn::Equals": [{"Ref": "RoleManagedPolicy5"},
""]}]},
   "HasPrivateKey": {"Fn::Not": [{"Fn::Equals": [{"Ref":
"IdentityProviderAddPrivateKey"}, ""]}]},
   "HasAssertionEncryptionMode": {"Fn::Not": [{"Fn::Equals": [{"Ref":
"IdentityProviderAssertionEncryptionMode"}, ""]}]}
 },
 "Resources": {
   "SAMLProvider": {
     "Type": "AWS::IAM::SAMLProvider",
     "Properties": {
       "Name": {"Fn::If": ["HasCustomProviderName", {"Ref":
"IdentityProviderName"}, {"Ref": "AWS::NoValue"}]},
       "SamlMetadataDocument": {"Ref": "IdentityProviderSAMLMetadataDocument"},
       "AddPrivateKey": {"Fn::If": ["HasPrivateKey", {"Ref":
"IdentityProviderAddPrivateKey"}, {"Ref": "AWS::NoValue"}]},
       "AssertionEncryptionMode": {"Fn::If": ["HasAssertionEncryptionMode", {"Ref":
"IdentityProviderAssertionEncryptionMode"}, {"Ref": "AWS::NoValue"}]}
     }
   },
   "SAMLFederatedRole": {
     "Type": "AWS::IAM::Role",
     "Properties": {
       "RoleName": {"Fn::If": ["HasCustomRoleName", {"Ref": "RoleName"}, {"Ref":
"AWS::NoValue"}]},
       "Path": {"Ref": "RolePath"},
       "Description": "SAML federated IAM role for SSO access with specified
permissions",
       "MaxSessionDuration": {"Ref": "RoleSessionDuration"},
       "PermissionsBoundary": {"Fn::If": ["HasPermissionsBoundary", {"Ref":
"RolePermissionsBoundary"}, {"Ref": "AWS::NoValue"}]},
```

```
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Federated": {"Ref": "SAMLProvider"}
              },
              "Action": [
                "sts:AssumeRole",
                "sts:SetSourceIdentity",
                "sts:TagSession"
              ],
              "Condition": {
                "StringEquals": {
                  "SAML:aud": "https://signin.aws.amazon.com/saml"
                }
              }
            }
          ]
        },
        "ManagedPolicyArns": {
          "Fn::Split": [
            ",",
            {
              "Fn::Join": [
                ",",
                [
                  {"Fn::If": ["HasPolicy1", {"Ref": "RoleManagedPolicy1"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy2", {"Ref": "RoleManagedPolicy2"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy3", {"Ref": "RoleManagedPolicy3"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy4", {"Ref": "RoleManagedPolicy4"}, {"Ref":
"AWS::NoValue"}]},
                  {"Fn::If": ["HasPolicy5", {"Ref": "RoleManagedPolicy5"}, {"Ref":
"AWS::NoValue"}]}
                ]
              ]
            }
          ]
        }
      }
```

```
    }
  },
  "Outputs": {
    "RoleARN": {
      "Description": "ARN of the created IAM Role",
      "Value": {"Fn::GetAtt": ["SAMLFederatedRole", "Arn"]},
      "Export": {
        "Name": {"Fn::Sub": "${AWS::StackName}-RoleARN"}
      }
    },
    "IdentityProviderARN": {
      "Description": "ARN of the created SAML Identity Provider",
      "Value": {"Ref": "SAMLProvider"},
      "Export": {
        "Name": {"Fn::Sub": "${AWS::StackName}-IdentityProviderARN"}
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] IAM: tutorial_saml-idp-and-federated-role'

Parameters:
  IdentityProviderName:
    Type: String
    Description: Name of the SAML Identity Provider (leave empty for auto-generated
 name like '{StackName}-{UniqueId}')
    Default: ""
    AllowedPattern: '^$|^[a-zA-Z0-9._-]+$'
    ConstraintDescription: Must be empty or contain only alphanumeric characters,
 periods, underscores, and hyphens

  IdentityProviderSAMLMetadataDocument:
    Type: String
    Description: SAML metadata document from identity provider

  IdentityProviderAddPrivateKey:
    Type: String
```

```
    Description: Optional private key for decrypting SAML assertions. The private
key must be a .pem file that uses AES-GCM or AES-CBC encryption algorithm to
decrypt SAML assertions.
    Default: ""

  IdentityProviderAssertionEncryptionMode:
    Type: String
    Description: Optional, sets encryption mode for SAML assertions
    Default: ""
    AllowedValues:
      - ""
      - "Allowed"
      - "Required"

  RoleName:
    Type: String
    Description: Name of the IAM Role (leave empty for auto-generated name like
'{StackName}-{UniqueId}')
    Default: ""
    AllowedPattern: '^$|^[\w+=,.@-]{1,64}$'
    ConstraintDescription: "Must be empty or 1-64 characters and can contain
alphanumeric characters and +=,.@-"

  RolePath:
    Type: String
    Description: Path for the IAM Role
    AllowedPattern: (^\/$)|(^\/.*\/$)
    Default: "/"

  RolePermissionsBoundary:
    Type: String
    Description: Optional ARN of the permissions boundary policy (leave empty for
none)
    Default: ""

  RoleSessionDuration:
    Description: The maximum session duration (in seconds) that you want to set for
the specified role (3600-43200)
    Type: Number
    MinValue: 3600
    MaxValue: 43200
    Default: 7200

  RoleManagedPolicy1:
```

```
    Type: String
    Description: Optional managed policy ARN 1
    Default: ""
  RoleManagedPolicy2:
    Type: String
    Description: Optional managed policy ARN 2
    Default: ""
  RoleManagedPolicy3:
    Type: String
    Description: Optional managed policy ARN 3
    Default: ""
  RoleManagedPolicy4:
    Type: String
    Description: Optional managed policy ARN 4
    Default: ""
  RoleManagedPolicy5:
    Type: String
    Description: Optional managed policy ARN 5
    Default: ""

Conditions:
  HasCustomProviderName: !Not [!Equals [!Ref IdentityProviderName, ""]]
  HasCustomRoleName: !Not [!Equals [!Ref RoleName, ""]]
  HasPermissionsBoundary: !Not [!Equals [!Ref RolePermissionsBoundary, ""]]
  HasPolicy1: !Not [!Equals [!Ref RoleManagedPolicy1, ""]]
  HasPolicy2: !Not [!Equals [!Ref RoleManagedPolicy2, ""]]
  HasPolicy3: !Not [!Equals [!Ref RoleManagedPolicy3, ""]]
  HasPolicy4: !Not [!Equals [!Ref RoleManagedPolicy4, ""]]
  HasPolicy5: !Not [!Equals [!Ref RoleManagedPolicy5, ""]]
  HasPrivateKey: !Not [!Equals [!Ref IdentityProviderAddPrivateKey, ""]]
  HasAssertionEncryptionMode: !Not [!Equals [!Ref
 IdentityProviderAssertionEncryptionMode, ""]]

Resources:
  SAMLProvider:
    Type: AWS::IAM::SAMLProvider
    Properties:
      Name: !If
        - HasCustomProviderName
        - !Ref IdentityProviderName
        - !Ref AWS::NoValue
      SamlMetadataDocument: !Ref IdentityProviderSAMLMetadataDocument
      AddPrivateKey: !If
        - HasPrivateKey
```

```
        - !Ref IdentityProviderAddPrivateKey
        - !Ref AWS::NoValue
      AssertionEncryptionMode: !If
        - HasAssertionEncryptionMode
        - !Ref IdentityProviderAssertionEncryptionMode
        - !Ref AWS::NoValue

  SAMLFederatedRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !If
        - HasCustomRoleName
        - !Ref RoleName
        - !Ref AWS::NoValue
      Path: !Ref RolePath
      Description: "SAML federated IAM role for SSO access with specified
permissions"
      MaxSessionDuration: !Ref RoleSessionDuration
      PermissionsBoundary: !If
        - HasPermissionsBoundary
        - !Ref RolePermissionsBoundary
        - !Ref AWS::NoValue
      AssumeRolePolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Principal:
              Federated: !Ref SAMLProvider
            Action:
              - 'sts:AssumeRole'
              - 'sts:SetSourceIdentity'
              - 'sts:TagSession'
            Condition:
              StringEquals:
                'SAML:aud': 'https://signin.aws.amazon.com/saml'
      ManagedPolicyArns: !Split
        - ','
        - !Join
          - ','
          - - !If [HasPolicy1, !Ref RoleManagedPolicy1, !Ref "AWS::NoValue"]
            - !If [HasPolicy2, !Ref RoleManagedPolicy2, !Ref "AWS::NoValue"]
            - !If [HasPolicy3, !Ref RoleManagedPolicy3, !Ref "AWS::NoValue"]
            - !If [HasPolicy4, !Ref RoleManagedPolicy4, !Ref "AWS::NoValue"]
            - !If [HasPolicy5, !Ref RoleManagedPolicy5, !Ref "AWS::NoValue"]
```

```
Outputs:
  RoleARN:
    Description: ARN of the created IAM Role
    Value: !GetAtt SAMLFederatedRole.Arn
    Export:
      Name: !Sub '${AWS::StackName}-RoleARN'

  IdentityProviderARN:
    Description: ARN of the created SAML Identity Provider
    Value: !Ref SAMLProvider
    Export:
      Name: !Sub '${AWS::StackName}-IdentityProviderARN'
```

IAM Identities

An IAM identity can be associated with one or more policies, which determine what actions an identity is authorized to perform, on which AWS resources, and under what conditions. IAM identities include IAM users, IAM groups, and IAM roles. An IAM entity is a type of identity that represents a human user or programmatic workload that can be authenticated and then authorized to perform actions in AWS accounts. IAM entities include IAM users and IAM roles. For definitions for commonly used terms, see Terms.

You can federate existing identities from an external identity provider. These identities will assume IAM roles to access AWS resources. For more information, see the section called “Identity providers and federation into AWS”.

You can also use AWS IAM Identity Center to create and manage identities and access to AWS resources. IAM Identity Center permission sets automatically create the IAM roles needed to provide access to resources. For more information, see What is IAM Identity Center?

You can use account access manager — an IAM feature that lets you assign existing IAM roles with the full IAM role feature set to IAM Identity Center users and groups. You can use it alongside permission sets or on its own.

The AWS account root user is an AWS account principal that is created when your AWS account is established. The root user has access to all AWS services and resources in the account. For more information, see the section called “IAM root user”.

Note

- Follow the Security best practices in IAM when working with IAM identities.
- Follow the root user best practices for your AWS account when working with the root user.
- If you're having trouble signing in, see Sign in to the AWS Management Console.

IAM root user

When you first create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user*. For more information, see AWS account root user.

IAM users

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. For more information, see IAM users.

IAM user groups

An *IAM user group* is an identity that specifies a collection of IAM users. For more information, see User groups.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It's similar to an IAM user, but isn't associated with a specific person. For more information, see IAM roles.

AWS account root user

When you first create an Amazon Web Services (AWS) account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user*. The email address and password that you used to create your AWS account are the credentials you use to sign in as your root user.

- Use the root user only to perform the tasks that require root-level permissions. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials.
- Follow the root user best practices for your AWS account.
- If you're having trouble signing in, see Sign in to the AWS Management Console.

> **Important**
>
> We strongly recommend that you don't use the root user for your everyday tasks and that you follow the root user best practices for your AWS account. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials.

While MFA is enforced for root users by default, it requires customer action to add MFA during the initial account creation or as prompted during sign-in. For more information about using MFA to protect the root user, see Multi-factor authentication for AWS account root user.

In addition to console access, you can manage programmatic access for the root user. You can Create access keys for the root user and Delete access keys for the root user when they are no longer needed.

Centrally manage root access for member accounts

To help you manage credentials at scale, you can centrally secure access to root user credentials for member accounts in AWS Organizations. When you enable AWS Organizations, you combine all your AWS accounts into an organization for central management. Centralizing root access lets you remove root user credentials and perform the following privileged tasks on member accounts.

Remove member account root user credentials

After you centralize root access for member accounts, you can choose to delete root user credentials from member accounts in your Organizations. You can remove the root user password, access keys, signing certificates, and deactivate multi-factor authentication (MFA). New accounts you create in Organizations have no root user credentials by default. Member accounts can't sign in to their root user or perform password recovery for their root user unless account recovery is enabled.

Perform privileged tasks that require root user credentials

Some tasks can only be performed when you sign in as the root user of an account. Some of these Tasks that require root user credentials can be performed by the management account or delegated administrator for IAM. To learn more about taking privileged actions on member accounts, see Perform a privileged task.

Enable account recovery of the root user

If you need to recover root user credentials for a member account, the Organizations management account or delegated administrator can perform the **Allow password recovery** privileged task. The person with access to the root user email inbox for the member account can reset the root user password to recover root user credentials. We recommend deleting root user credentials after you complete the task that requires access to the root user.

Centralize root access for member accounts

Root user credentials are the initial credentials assigned to each AWS account that has complete access to all AWS services and resources in the account. When you enable AWS Organizations, you combine all your AWS accounts into an organization for central management. Each member account has its own root user with default permissions to perform any action in the member account. We recommend you centrally secure the root user credentials of AWS accounts managed using AWS Organizations to prevent root user credential recovery and access at scale.

After you centralize root access, you can choose to delete root user credentials from member accounts in your organization. You can remove the root user password, access keys, signing certificates, and deactivate multi-factor authentication (MFA). New accounts you create in AWS Organizations have no root user credentials by default. Member accounts can't sign in to their root user or perform password recovery for their root user.

Note

While some Tasks that require root user credentials can be performed by the management account or delegated administrator for IAM, some tasks can only be performed when you sign in as the root user of an account.
If you need to recover root user credentials for a member account to perform one of these tasks, follow the steps in Perform a privileged task and select **Allow password recovery**. The person with access to the root user email inbox for the member account can then follow the steps to reset the root user password and sign in to the member account root user.
We recommend deleting root user credentials once you complete the task that requires access to the root user.

Prerequisites

Before you centralize root access, you must have an account configured with the following settings:

- You must have the following IAM permissions:
  - `iam:GetAccessKeyLastUsed`
  - `iam:GetAccountSummary`
  - `iam:GetLoginProfile`
  - `iam:GetUser`
  - `iam:ListAccessKeys`
  - `iam:ListMFADevices`
  - `iam:ListSigningCertificates`
  - `sts:AssumeRoot`

  > **Note**
  >
  > To audit the root user credential status of a member account, you can use the IAMAuditRootUserCredentials AWS managed policy to scope down permissions when you perform a privileged task on an AWS Organizations member account, or use any policy with access to `iam:GetAccountSummary`.
  > To generate the root user credential information report, other policies only need the `iam:GetAccountSummary` action to produce the same output. You can also list or get individual root user credential information, including:
  >
  > - Whether a root user password is present
  > - Whether a root user access key is present and when it was last used
  > - Whether the root user has associated signing certificates
  > - Root user associated MFA devices
  > - List of the consolidated root user credential status

- You must manage your AWS accounts in AWS Organizations.
- You must have the following permissions to enable this feature in your organization:
  - `iam:EnableOrganizationsRootCredentialsManagement`
  - `iam:EnableOrganizationsRootSessions`
  - `iam:ListOrganizationsFeatures`

- `organizations:EnableAwsServiceAccess`
- `organizations:ListAccountsForParent`
- `organizations:RegisterDelegatedAdministrator`

- To ensure optimal console functionality, we recommend enabling the following additional permissions:
  - `organizations:DescribeAccount`
  - `organizations:DescribeOrganization`
  - `organizations:ListAWSServiceAccessForOrganization`
  - `organizations:ListDelegatedAdministrators`
  - `organizations:ListOrganizationalUnitsForParent`
  - `organizations:ListParents`
  - `organizations:ListTagsForResource`

Enabling centralized root access (console)

To enable this feature for member accounts in the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Root access management**, and then select **Enable**.

   > **Note**
   >
   > If you see **Root access management is disabled**, enable trusted access for AWS Identity and Access Management in AWS Organizations. For details, see AWS IAM and AWS Organizations in the *AWS Organizations User Guide*.

3. In the Capabilities to enable section, choose which features to enable.
   - Select **Root credentials management** to allow the management account and the delegated administrator for IAM to delete root user credentials for member accounts. You must enable Privileged root actions in member accounts to allow member accounts to recover their root user credentials after they have been deleted.

- Select **Privileged root actions in member accounts** to allow the management account and the delegated administrator for IAM to perform certain tasks that require root user credentials.

4. (Optional) Enter the account ID of the **Delegated administrator** that is authorized to manage root user access and take privileged actions on member accounts. We recommend an account that is intended for security or management purposes.
5. Choose **Enable**.

Enabling centralized root access (AWS CLI)

To enable centralized root access from the AWS Command Line Interface (AWS CLI)

1. If you haven't already enabled trusted access for AWS Identity and Access Management in AWS Organizations, use the following command: aws organizations enable-aws-service-access.
2. Use the following command to allow the management account and the delegated administrator to delete root user credentials for member accounts: aws iam enable-organizations-root-credentials-management.
3. Use the following command to allow the management account and the delegated administrator to perform certain tasks that require root user credentials: aws iam enable-organizations-root-sessions.
4. (Optional) Use the following command to register a delegated administrator: aws organizations register-delegated-administrator.

   The following example assigns account 111111111111 as the delegated administrator for the IAM service.

   ```
   aws organizations register-delegated-administrator
   --service-principal iam.amazonaws.com
   --account-id 111111111111
   ```

Enabling centralized root access (AWS API)

To enable centralized root access from the AWS API

1. If you haven't already enabled trusted access for AWS Identity and Access Management in AWS Organizations, use the following command: EnableAWSServiceAccess.

2. Use the following command to allow the management account and the delegated administrator to delete root user credentials for member accounts: EnableOrganizationsRootCredentialsManagement.
3. Use the following command to allow the management account and the delegated administrator to perform certain tasks that require root user credentials: EnableOrganizationsRootSessions.
4. (Optional) Use the following command to register a delegated administrator: RegisterDelegatedAdministrator.

Next steps

Once you've centrally secured privileged credentials for the member accounts in your organization, see Perform a privileged task to take privileged actions on a member account.

Perform a privileged task on an AWS Organizations member account

The AWS Organizations management account or a delegated administrator account for IAM can perform some privileged tasks on member accounts that would otherwise require root user credentials. With centralized root access, these tasks are performed through short-term privileged sessions. These sessions provide temporary credentials scoped to specific privileged actions, without requiring root user sign-in on the member account.

Once you launch a privileged session, you can delete a misconfigured Amazon S3 bucket policy, delete a misconfigured Amazon SQS queue policy, delete the root user credentials for a member account, and reenable root user credentials for a member account.

> **Note**
>
> To use centralized root access, you must sign in via a management account or a delegated administrator account as an IAM user or role with the `sts:AssumeRoot` permission explicitly granted. You cannot use root user credentials to call `sts:AssumeRoot`.

Prerequisites

Before you can launch a privileged session, you must have the following settings:

- You have enabled centralized root access in your organization. For steps to enable this feature, see Centralize root access for member accounts.
- Your management account or delegated administrator account has the following permissions: `sts:AssumeRoot`

Taking a privileged action on a member account (console)

To launch a session for privileged action in a member account in the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Root access management**.
3. Select a name from the member account list, and choose **Take privileged action**.
4. Choose the privileged action you want to take in the member account.
   - Select **Delete Amazon S3 bucket policy** to remove a misconfigured bucket policy that denies all principals from accessing the Amazon S3 bucket.
     a. Choose **Browse S3** to select a name from the buckets owned by the member account, and select **Choose**.
     b. Choose **Delete bucket policy**.
     c. Use the Amazon S3 console to correct the bucket policy after deleting the misconfigured policy. For more information, see Adding a bucket policy by using the Amazon S3 console in the *Amazon S3 User Guide*.
   - Select **Delete Amazon SQS policy** to delete an Amazon Simple Queue Service resource-based policy that denies all principals from accessing an Amazon SQS queue.
     a. Enter the queue name in **SQS queue name**, and select **Delete SQS policy**.
     b. Use the Amazon SQS console to correct the queue policy after deleting the misconfigured policy. For more information, see Configuring an access policy in Amazon SQS in the *Amazon SQS Developer Guide*.
   - Select **Delete root credentials** to remove root access from a member account. Deleting root user credentials removes the root user password, access keys, signing certificates, and deactivates multi-factor authentication (MFA) for the member account.
     - Choose **Delete root credentials**.

- Select **Allow password recovery** to recover root user credentials for a member account.

  This option is only available when the member account has no root user credentials.

  a. Choose **Allow password recovery**.

  b. After taking this privileged action, the person with access to the root user email inbox for the member account can reset the root user password and sign in to the member account root user.

Taking a privileged action on a member account (AWS CLI)

To launch a session for privileged action in a member account from the AWS Command Line Interface

1. Use the following command to assume a root user session: aws sts assume-root.

> **Note**
>
> The global endpoint is not supported for `sts:AssumeRoot`. You must send this request to a Regional AWS STS endpoint. For more information, see Manage AWS STS in an AWS Region.

When you launch a privileged root user session for a member account, you must define `task-policy-arn` to scope the session to the privileged action to be performed during the session. You can use one of following AWS managed policies to scope privileged session actions.

- IAMAuditRootUserCredentials
- IAMCreateRootUserPassword
- IAMDeleteRootUserCredentials
- S3UnlockBucketPolicy
- SQSUnlockQueuePolicy

To limit the actions a management account or delegated administrator can perform during a privileged root user session, you can use the AWS STS condition key sts:TaskPolicyArn.

In the following example, the delegated administrator assumes root to delete the root user credentials for the member account ID *111122223333*.

```
aws sts assume-root \
  --target-principal 111122223333 \
  --task-policy-arn arn=arn:aws:iam::aws:policy/root-task/
IAMDeleteRootUserCredentials \
  --duration-seconds 900
```

2. Use the `SessionToken`, `AccessKeyId`, and `SecretAccessKey` from the response to perform privileged actions in the member account. You can omit the user name and password in the request to default to the member account.

   - **Check the status of root user credentials**. Use the following commands to check the status of root user credentials for a member account.

     - get-user
     - get-login-profile
     - list-access-keys
     - list-signing-certificates
     - list-mfa-devices
     - get-access-key-last-used

   - **Delete root user credentials**. Use the following commands to delete root access. You can remove the root user password, access keys, signing certificates, and deactivate multi-factor authentication (MFA) to remove all access to and recovery of the root user.

     - delete-login-profile
     - delete-access-key
     - delete-signing-certificate
     - deactivate-mfa-device

   - **Delete Amazon S3 bucket policy**. Use the following commands to read, edit, and delete a misconfigured bucket policy that denies all principals from accessing the Amazon S3 bucket.

     - list-buckets
     - get-bucket-policy

    - put-bucket-policy
    - delete-bucket-policy
- **Delete Amazon SQS policy**. Use the following commands to view and delete an Amazon Simple Queue Service resource-based policy that denies all principals from accessing an Amazon SQS queue.
    - list-queues
    - get-queue-url
    - get-queue-attributes
    - set-queue-attributes
- **Allow password recovery**. Use the following commands to view the user name and recover root user credentials for a member account.
    - get-login-profile
    - create-login-profile

Taking a privileged action on a member account (AWS API)

To launch a session for privileged action in a member account from the AWS API

1. Use the following command to assume a root user session: AssumeRoot.

> **Note**
>
> The global endpoint is not supported for AssumeRoot. You must send this request to a Regional AWS STS endpoint. For more information, see Manage AWS STS in an AWS Region.

When you launch a privileged root user session for a member account, you must define `TaskPolicyArn` to scope the session to the privileged action to be performed during the session. You can use one of following AWS managed policies to scope privileged session actions.

- IAMAuditRootUserCredentials
- IAMCreateRootUserPassword

- IAMDeleteRootUserCredentials
- S3UnlockBucketPolicy
- SQSUnlockQueuePolicy

To limit the actions a management account or delegated administrator can perform during a privileged root user session, you can use the AWS STS condition key sts:TaskPolicyArn.

In the following example, the delegated administrator assumes root to read, edit and delete a misconfigured resource-based policy for an Amazon S3 bucket for the member account ID *111122223333*.

```
https://sts.us-east-2.amazonaws.com/
  ?Version=2011-06-15
  &Action=AssumeRoot
  &TargetPrincipal=111122223333
  &PolicyArns.arn=arn:aws:iam::aws:policy/root-task/S3UnlockBucketPolicy
  &DurationSeconds=900
```

2. Use the `SessionToken`, `AccessKeyId`, and `SecretAccessKey` from the response to perform privileged actions in the member account. You can omit the user name and password in the request to default to the member account.
   - **Check the status of root user credentials**. Use the following commands to check the status of root user credentials for a member account.
     - GetUser
     - GetLoginProfile
     - ListAccessKeys
     - ListSigningCertificates
     - ListMFADevices
     - GetAccessKeyLastUsed
   - **Delete root user credentials**. Use the following commands to delete root access. You can remove the root user password, access keys, signing certificates, and deactivate multi-factor authentication (MFA) to remove all access to and recovery of the root user.
     - DeleteLoginProfile
     - DeleteAccessKey

    - DeleteSigningCertificate
    - DeactivateMfaDevice
- **Delete Amazon S3 bucket policy**. Use the following commands to read, edit, and delete a misconfigured bucket policy that denies all principals from accessing the Amazon S3 bucket.
    - ListBuckets
    - GetBucketPolicy
    - PutBucketPolicy
    - DeleteBucketPolicy
- **Delete Amazon SQS policy**. Use the following commands to view and delete an Amazon Simple Queue Service resource-based policy that denies all principals from accessing an Amazon SQS queue.
    - ListQueues
    - GetQueueUrl
    - GetQueueAttributes
    - SetQueueAttributes
- **Allow password recovery**. Use the following commands to view the user name and recover root user credentials for a member account.
    - GetLoginProfile
    - CreateLoginProfile

Multi-factor authentication for AWS account root user

> **Important**
>
> AWS recommends that you use a passkey or security key for MFA to AWS, wherever possible as they are more resistant to attacks such as phishing. For more information, see Passkeys and security keys.

Multi-factor authentication (MFA) is a simple and effective mechanism to enhance your security. The first factor — your password — is a secret that you memorize, also known as a knowledge

factor. Other factors can be possession factors (something you have, such as a security key) or inherence factors (something you are, such as a biometric scan). For increased security, we strongly recommend that you configure multi-factor authentication (MFA) to help protect your AWS resources.

> **Note**
>
> All AWS account types (standalone, management, and member accounts) require MFA to be configured for their root user. Users must register MFA within 14 days of their first sign-in attempt to access the AWS Management Console if MFA is not already enabled.

You can enable MFA for the AWS account root user and IAM users. When you enable MFA for the root user, it only affects the root user credentials. For more information about how to enable MFA for your IAM users, see AWS Multi-factor authentication in IAM.

> **Note**
>
> AWS accounts managed using AWS Organizations might have the option to centrally manage root access for member accounts to prevent credential recovery and access at scale. If this option is enabled, you can delete root user credentials from member accounts, including passwords and MFA, effectively preventing sign-in as the root user, password recovery, or setting up MFA. Alternatively, if you prefer to maintain password-based sign-in methods, secure your account by registering MFA to enhance account protection.

Before you enable MFA for your root user, review and update your account settings and contact information to make sure that you have access to the email and phone number. If your MFA device is lost, stolen, or not working, you can still sign in as the root user by verifying your identity using that email and phone number. To learn about signing in using alternative factors of authentication, see Recover an MFA protected identity in IAM. To disable this feature, contact AWS Support.

AWS supports the following MFA types for your root user:

- Passkeys and security keys
- Virtual authenticator applications
- Hardware TOTP tokens

Passkeys and security keys

AWS Identity and Access Management supports passkeys and security keys for MFA. Based on FIDO standards, passkeys use public key cryptography to provide strong, phishing-resistant authentication that is more secure than passwords. AWS supports two types of passkeys: device-bound passkeys (security keys) and synced passkeys.

- **Security keys**: These are physical devices, like a YubiKey, used as a second factor for authentication. A single security key can support multiple root user accounts and IAM users.
- **Synced passkeys**: These use credential managers from providers such as Google, Apple, Microsoft accounts, and third-party services like 1Password, Dashlane, and Bitwarden as a second factor.

You can use built-in biometric authenticators, like Touch ID on Apple MacBooks, to unlock your credential manager and sign in to AWS. Passkeys are created with your chosen provider using your fingerprint, face, or device PIN. You can also use a cross-device authentication (CDA) passkey from one device, like a mobile device or hardware security key, to sign in on another device like a laptop. For more information, see cross-device authentication (CDA).

You can sync passkeys across your devices to facilitate sign-ins with AWS, enhancing usability and recoverability. For more information about enabling passkeys and security keys, see Enable a passkey or security key for the root user (console).

The FIDO Alliance maintains a list of all FIDO Certified products that are compatible with FIDO specifications.

Virtual authenticator applications

A virtual authenticator application runs on a phone or other device and emulates a physical device. Virtual authenticator apps implement the time-based one-time password (TOTP) algorithm and support multiple tokens on a single device. The user must type a valid code from the device when prompted during sign-in. Each token assigned to a user must be unique. A user can't type a code from another user's token to authenticate.

We do recommend that you use a virtual MFA device while waiting for hardware purchase approval or while you wait for your hardware to arrive. For a list of a few supported apps that you can use as virtual MFA devices, see Multi-Factor Authentication (MFA). For instructions on setting up a virtual MFA device with AWS, see Enable a virtual MFA device for the root user (console).

Hardware TOTP tokens

A hardware device generates a six-digit numeric code based on the time-based one-time password (TOTP) algorithm. The user must type a valid code from the device on a second webpage during sign-in. Each MFA device assigned to a user must be unique. A user cannot type a code from another user's device to be authenticated. For information on supported hardware MFA devices, see Multi-Factor Authentication (MFA). For instructions on setting up a hardware TOTP token with AWS, see Enable a hardware TOTP token for the root user (console).

If you want to use a physical MFA device, we recommend that you use FIDO security keys as an alternative to hardware TOTP devices. FIDO security keys offer the benefits of no battery requirements, phishing resistance, and they support multiple root and IAM users on a single device for enhanced security.

Topics

- Enable a passkey or security key for the root user (console)
- Enable a virtual MFA device for the root user (console)
- Enable a hardware TOTP token for the root user (console)

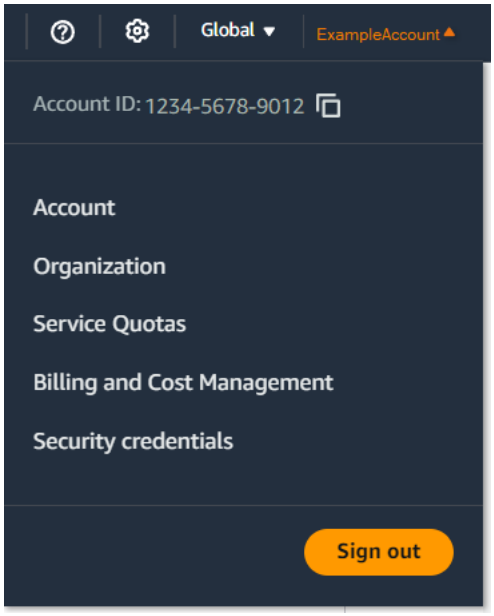
Enable a passkey or security key for the root user (console)

You can configure and enable a passkey for your root user from the AWS Management Console only, not from the AWS CLI or AWS API.

To enable a passkey or security key for your root user (console)

1. Open the AWS Management Console and sign in using your root user credentials.

   For instructions, see Sign in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.
2. On the right side of the navigation bar, choose your account name, and then choose **Security credentials**.



3. On your root user **My security credentials** page, under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
4. On the **MFA device name** page, enter a **Device name**, choose **Passkey or Security Key**, and then choose **Next**.
5. On **Set up device**, set up your passkey. Create a passkey with biometric data like your face or fingerprint, with a device pin, or by inserting the FIDO security key into your computer's USB port and tapping it.
6. Follow the instructions on your browser to choose a passkey provider or where you want to store your passkey to use across your devices.
7. Choose **Continue**.

You have now registered your passkey for use with AWS. The next time you use your root user credentials to sign in, you must authenticate with your passkey to complete the sign-in process.

For help troubleshooting issues with your FIDO security key, see Troubleshoot Passkeys and FIDO Security Keys.

Enable a virtual MFA device for the root user (console)

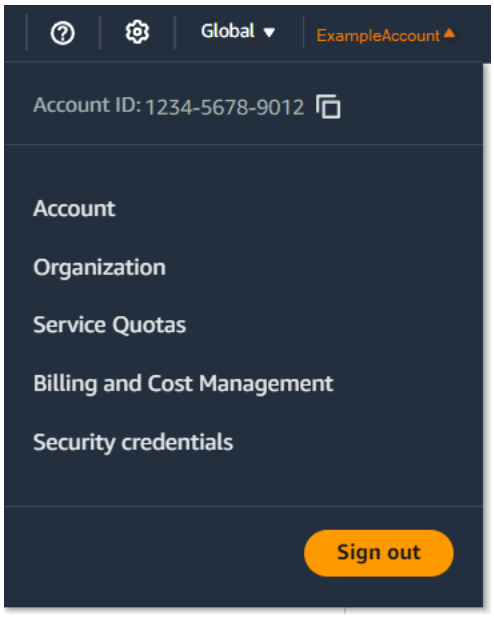
You can use the AWS Management Console to configure and enable a virtual MFA device for your root user. To enable MFA devices for the AWS account, you must be signed in to AWS using your root user credentials.

To configure and enable a virtual MFA device for use with your root user (console)

1. Open the AWS Management Console and sign in using your root user credentials.

   For instructions, see Sign in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.

2. On the right side of the navigation bar, choose your account name, and choose **Security credentials**.



3. In the **Multi-Factor Authentication (MFA)** section, choose **Assign MFA device**.
4. In the wizard, type a **Device name**, choose **Authenticator app**, and then choose **Next**.

   IAM generates and displays configuration information for the virtual MFA device, including a QR code graphic. The graphic is a representation of the secret configuration key that is available for manual entry on devices that do not support QR codes.

5. Open the virtual MFA app on the device.

   If the virtual MFA app supports multiple virtual MFA devices or accounts, choose the option to create a new virtual MFA device or account.

6. The easiest way to configure the app is to use the app to scan the QR code. If you cannot scan the code, you can type the configuration information manually. The QR code and secret configuration key generated by IAM are tied to your AWS account and cannot be used with a different account. They can, however, be reused to configure a new MFA device for your account in case you lose access to the original MFA device.

- To use the QR code to configure the virtual MFA device, from the wizard, choose **Show QR code**. Then follow the app instructions for scanning the code. For example, you might need to choose the camera icon or choose a command like **Scan account barcode**, and then use the device's camera to scan the QR code.
- In the **Set up device** wizard, choose **Show secret key**, and then type the secret key into your MFA app.

> **Important**
>
> Make a secure backup of the QR code or secret configuration key, or make sure that you enable multiple MFA devices for your account. You can register up to **eight** MFA devices of any combination of the currently supported MFA types with your AWS account root user and IAM users. A virtual MFA device might become unavailable, for example, if you lose the smartphone where the virtual MFA device is hosted. If that happens and you are not able to sign in to your account with no additional MFA devices attached to the user or even by Recovering a root user MFA device, you will not be able to sign in to your account and you will have to contact customer service to remove MFA protection for the account.

The device starts generating six-digit numbers.

7. In the wizard, in the **MFA code 1** box, type the one-time password that currently appears in the virtual MFA device. Wait up to 30 seconds for the device to generate a new one-time password. Then type the second one-time password into the **MFA code 2** box. Choose **Add MFA**.

> **Important**
>
> Submit your request immediately after generating the code. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resync the device.

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see MFA enabled sign-in.

Enable a hardware TOTP token for the root user (console)

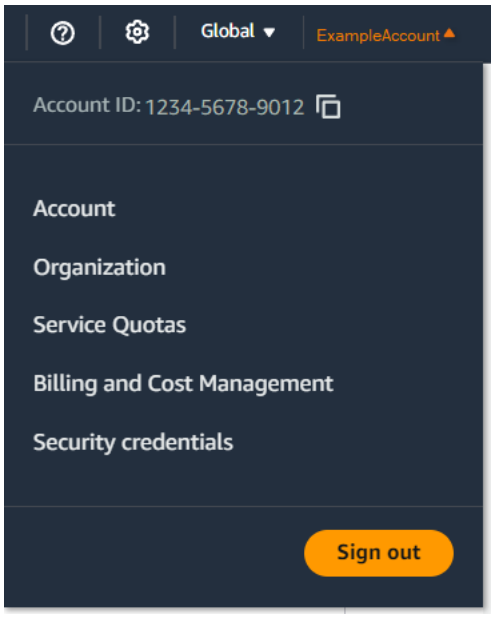
You can configure and enable a physical MFA device for your root user from the AWS Management Console only, not from the AWS CLI or AWS API.

> **Note**
>
> You might see different text, such as **Sign in using MFA** and **Troubleshoot your authentication device**. However, the same features are provided. In either case, if you cannot verify your account email address and phone number using alternative factors of authentication, contact AWS Support to delete your MFA setting.

To enable a hardware TOTP token for your root user (console)

1. Open the AWS Management Console and sign in using your root user credentials.

   For instructions, see Sign in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.
2. On the right side of the navigation bar, choose your account name, and then choose **Security credentials**.



3. Expand the **Multi-factor authentication (MFA)** section.

4. Choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Hardware TOTP token**, and then choose **Next**.
6. In the **Serial number** box, type the serial number that is found on the back of the MFA device.
7. In the **MFA code 1** box, type the six-digit number displayed by the MFA device. You might need to press the button on the front of the device to display the number.



8. Wait 30 seconds while the device refreshes the code, and then type the next six-digit number into the **MFA code 2** box. You might need to press the button on the front of the device again to display the second number.
9. Choose **Add MFA**. The MFA device is now associated with the AWS account.

> **Important**
>
> Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resync the device.

The next time you use your root user credentials to sign in, you must type a code from the MFA device.

Change the password for the AWS account root user

You can change the email address and password from either the Security Credentials or the **Account** page. You can also choose **Forgot password?** on the AWS sign-in page to reset your password.

To change the root user's password, you must sign in as the AWS account root user and not as an IAM user. To learn how to reset a *forgotten* root user password, see Reset a lost or forgotten root user password.

To protect your password, it's important to follow these best practices:

- Change your password periodically.
- Keep your password private because anyone who knows your password can access your account.
- Use a different password on AWS than you use on other sites.
- Avoid passwords that are easy to guess. These include passwords such as `secret`, `password`, `amazon`, or `123456`. Also avoid things like dictionary words, your name, email address, or other personal information that someone can easily obtain.

Important

AWS accounts managed using AWS Organizations might have centralized root access enabled for member accounts. These member accounts do not have root user credentials, can't sign in as a root user, and are prevented from recovering the root user password. Contact your administrator if you need to perform a task that requires root user credentials.

AWS Management Console

To change the password for the root user

Minimum permissions

To perform the following steps, you must have at least the following IAM permissions:

- You must sign in as the AWS account root user, which requires no additional AWS Identity and Access Management (IAM) permissions. You can't perform these steps as an IAM user or role.

1. Open the AWS Management Console and sign in using your root user credentials.

   For instructions, see Sign in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.
2. In the upper right corner of the console, choose your account name or number, and then select **Security Credentials**.
3. On the **Account** page, next to **Account settings**, choose **Edit**. You are prompted to re-authenticate for security purposes.

> **Note**
>
> If you don't see the **Edit** option, it is likely that you are not signed in as the root user for your account. You can't modify account settings while signed in as an IAM user or role.

4. On the **Update account settings** page, under **Password**, choose **Edit**.
5. On the **Update your password** page, fill out the fields for **Current password**, **New password**, and **Confirm new password**.

> **Important**
>
> Make sure to choose a strong password. Although you can set an account password policy for IAM users, that policy doesn't apply to the root user.

AWS requires that your password meet the following conditions:

- It must have a minimum of 8 characters and a maximum of 128 characters.
- It must include a minimum of three of the following mix of character types: uppercase, lowercase, numbers, and ! @ # $ % ^ & * () <> [] {} | _+-= symbols.
- It must not be identical to your AWS account name or email address.

6. Choose **Save changes**.

AWS CLI or AWS SDK

This task isn't supported in the AWS CLI or by an API operation from one of the AWS SDKs. You can perform this task only by using the AWS Management Console.

Reset a lost or forgotten root user password

When you first created your AWS account, you provided an email address and password. These are your AWS account root user credentials. If you forget your root user password, you can reset the password from the AWS Management Console.

AWS accounts managed using AWS Organizations might have centralized root access enabled for member accounts. These member accounts do not have root user credentials, can't sign in as a root user, and are prevented from recovering the root user password. Contact your administrator if you need to perform a task that requires root user credentials.

> **Important**
>
> **Having trouble signing in to AWS?** Make sure that you're on the correct AWS sign-in page for your type of user. If you are the AWS account root user (account owner), you can sign in to AWS using the credentials that you set up when you created the AWS account. If you are an IAM user, your account administrator can give you the credentials that you can use to sign in to AWS. If you need to request support, do not use the feedback link on this page, as the form is received by the AWS Documentation team, not Support. Instead, on the Contact Us page choose **Still unable to log into your AWS account** and then choose one of the available support options.

To reset your root user password

1. Open the AWS Management Console and sign in using your root user credentials.

   For instructions, see Sign in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.

   > **Note**
   >
   > If you are signed in to the AWS Management Console with *IAM user* credentials, then you must sign out before you can reset the root user password. If you see the account-specific IAM user sign-in page, choose **Sign-in using root account credentials** near the bottom of the page. If necessary, provide your account email address and choose **Next** to access the **Root user sign in** page.

2. Choose **Forgot your password?**.

   > **Note**
   >
   > If you are an IAM user, this option is not available. The **Forgot your password?** option is only available for the root user account. IAM users must ask their administrator to reset a forgotten password. For more information, see I forgot my IAM user password

for my AWS account. If you sign in through the AWS access portal, see Resetting your IAM Identity Center user password.

3. Provide the email address that is associated with the account. Then provide the CAPTCHA text and choose **Continue**.
4. Check the email that is associated with your AWS account for a message from Amazon Web Services. The email will come from an address ending in `@verify.signin.aws`. Follow the directions in the email. If you don't see the email in your account, check your spam folder. If you no longer have access to the email, see I don't have access to the email for my AWS account in the *AWS Sign-In User Guide*.

Create access keys for the root user

Warning

We strongly recommend that you do **not** create access key pairs for your root user. Because only a few tasks require the root user and you typically perform those tasks infrequently, we recommend signing in to the AWS Management Console to perform the root user tasks. Before creating access keys, review the alternatives to long-term access keys.

Although we don't recommend it, you can create access keys for your root user so that you can run commands in the AWS Command Line Interface (AWS CLI) or use API operations from one of the AWS SDKs using root user credentials. When you create access keys, you create the access key ID and secret access key as a set. During access key creation, AWS gives you one opportunity to view and download the secret access key part of the access key. If you don't download it or if you lose it, you can delete the access key and then create a new one. You can create root user access keys with the console, AWS CLI, or AWS API.

A newly created access key has the status of *active*, which means that you can use the access key for CLI and API calls. You can assign up to two access keys to the root user.

Access keys that are not in use should be inactivated. After an access key is inactive, you can't use it for API calls. Inactive keys still count toward your limit. You can create or delete an access key any time. However, when you delete an access key, it's gone forever and can't be retrieved.

AWS Management Console

To create an access key for the AWS account root user

> **Minimum permissions**
>
> To perform the following steps, you must have at least the following IAM permissions:
>
> - You must sign in as the AWS account root user, which requires no additional AWS Identity and Access Management (IAM) permissions. You can't perform these steps as an IAM user or role.

1. Open the AWS Management Console and sign in using your root user credentials.

   For instructions, see Sign in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.
2. In the upper right corner of the console, choose your account name or number and then choose **Security Credentials**.
3. In the **Access keys** section, choose **Create access key**. If this option is not available, then you already have the maximum number of access keys. You must delete one of the existing access keys before you can create a new key. For more information, see IAM Object Quotas.
4. On the **Alternatives to root user access keys** page, review the security recommendations. To continue, select the checkbox, and then choose **Create access key**.
5. On the **Retrieve access key** page, your **Access key** ID is displayed.
6. Under **Secret access key**, choose **Show** and then copy the access key ID and secret key from your browser window and paste it somewhere secure. Alternatively, you can choose **Download .csv file** which will download a file named `rootkey.csv` that contains the access key ID and the secret key. Save the file somewhere safe.
7. Choose **Done**. When you no longer need the access key we recommend that you delete it, or at least consider deactivating it so that no one can misuse it.

AWS CLI & SDKs

To create an access key for the root user

> **Note**
>
> To run the following command or API operation as the root user, you must already have one active access key pair. If you don't have any access keys, create the first access key using the AWS Management Console. Then, you can use the credentials from that first access key with the AWS CLI to create the second access key, or to delete an access key.

- AWS CLI: aws iam create-access-key

  Example

```
$ aws iam create-access-key
{
    "AccessKey": {
        "UserName": "MyUserName",
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "Status": "Active",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "CreateDate": "2021-04-08T19:30:16+00:00"
    }
}
```

- AWS API: CreateAccessKey in the *IAM API Reference*.

Delete access keys for the root user

You can use the AWS Management Console, the AWS CLI or the AWS API to delete the root user access keys.

AWS Management Console

To delete an access key for the root user

> **Minimum permissions**
>
> To perform the following steps, you must have at least the following IAM permissions:

- You must sign in as the AWS account root user, which requires no additional AWS Identity and Access Management (IAM) permissions. You can't perform these steps as an IAM user or role.

1. Open the AWS Management Console and sign in using your root user credentials.

   For instructions, see Sign in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.
2. In the upper right corner of the console, choose your account name or number and then choose **Security Credentials**.
3. In the **Access keys** section, select the access key that you want to delete, and then, under **Actions**, choose **Delete**.

   Note

   Alternatively, you can **Deactivate** an access key, instead of permanently deleting it. This way you can resume using it in the future without having to change either the key ID or secret key. While the key is inactive, any attempts to use it in requests to the AWS API fail with the error access denied.
4. On the **Delete <access key ID>** dialog box, choose **Deactivate**, enter the access key ID to confirm you want to delete it, and then choose **Delete**.

AWS CLI & SDKs

To delete an access key for the root user

Minimum permissions

To perform the following steps, you must have at least the following IAM permissions:

- You must sign in as the AWS account root user, which requires no additional AWS Identity and Access Management (IAM) permissions. You can't perform these steps as an IAM user or role.

- AWS CLI: aws iam delete-access-key

  Example

  ```
  $ aws iam delete-access-key \
      --access-key-id AKIAIOSFODNN7EXAMPLE
  ```

  This command produces no output when successful.

- AWS API: DeleteAccessKey

Tasks that require root user credentials

We recommend that you configure an administrative user in AWS IAM Identity Center to perform daily tasks and access AWS resources. However, you can perform the tasks listed below only when you sign in as the root user of an account.

To simplify managing privileged root user credentials across member accounts in AWS Organizations, you can enable centralized root access to help you centrally secure highly privileged access to your AWS accounts. Centrally manage root access for member accounts lets you centrally remove and prevent long-term root user credential recovery, improving account security in your organization. After you enable this feature, you can perform the following privileged tasks on member accounts.

- Remove member account root user credentials to prevent account recovery of the root user. You can also allow password recovery to recover root user credentials for a member account.
- Remove a misconfigured bucket policy that denies all principals from accessing an Amazon S3 bucket.
- Delete an Amazon Simple Queue Service resource-based policy that denies all principals from accessing an Amazon SQS queue.

Account Management Tasks

- Change your AWS account settings. Standalone AWS accounts that are not part of AWS Organizations require root credentials to update the email address, root user password, and root user access keys. Other account settings, such as account name, contact information, alternate contacts, payment currency preference, and AWS Regions, don't require root user credentials.

> **Note**
>
> AWS Organizations, with all features enabled, can be used to manage member account settings centrally from the management account and delegated admin accounts. Authorized IAM users or IAM roles in both the management account and delegated admin accounts can close member accounts and update the root email addresses, account names, contact information, alternate contacts, and AWS Regions of member accounts.

- Close your AWS account. Standalone AWS accounts that are not part of AWS Organizations require root credentials to close the account. With AWS Organizations, you can close the member accounts centrally from the management account and delegated admin accounts.
- Reopen your AWS account. If you closed a standalone account yourself and it has no outstanding balance, you can reopen it during the post-closure period. You must sign in as the root user to do so.
- Restore IAM user permissions. If the only IAM administrator accidentally revokes their own permissions, you can sign in as the root user to edit policies and restore those permissions.

Billing Tasks

- Activate IAM access to the Billing and Cost Management console.
- Some Billing tasks are limited to the root user. See Managing an AWS account in AWS Billing User Guide for more information.
- View certain tax invoices. An IAM user with the aws-portal:ViewBilling permission can view and download VAT invoices from AWS Europe, but not AWS Inc. or Amazon Internet Services Private Limited (AISPL).

AWS GovCloud (US) Tasks

- Sign up for AWS GovCloud (US).
- Request AWS GovCloud (US) account root user access keys from AWS Support.

Amazon EC2 Task

- Register as a seller in the Reserved Instance Marketplace.

AWS KMS Task

- In the event that an AWS Key Management Service key becomes unmanageable, an administrator can recover it by contacting Support; however, Support responds to your root user's primary phone number for authorization by confirming the ticket OTP.

Amazon Simple Storage Service Tasks

- Configure an Amazon S3 bucket to enable MFA (multi-factor authentication).
- Edit or delete an Amazon S3 bucket policy that denies all principals.

  You can use privileged actions to unlock an Amazon S3 bucket with a misconfigured bucket policy. For details, see Perform a privileged task on an AWS Organizations member account.

Amazon Simple Queue Service Task

- Edit or delete an Amazon SQS resource-based policy that denies all principals.

  You can use privileged actions to unlock an Amazon SQS queue with a misconfigured resource-based policy. For details, see Perform a privileged task on an AWS Organizations member account.

Additional resources

For more information about the AWS root user, see the following resources:

- For help with root user issues, see Troubleshoot issues with the root user.
- To centrally manage root user email addresses in AWS Organizations, see Updating the root user email address for a member account in the *AWS Organizations User Guide*.

The following articles provide additional information about working with the root user.

- What are some best practices for securing my AWS account and its resources?
- How can I create an EventBridge event rule to notify me that my root user was used?
- Monitor and notify on AWS account root user activity
- Monitor IAM root user activity

IAM users

> **Important**
>
> IAM best practices recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials. We recommend that you only use IAM users for specific use cases not supported by federated users.
> If you're using our new AWS experience and want to provide human users with access to your projects or AWS Settings, you must add them to your team. In our new experience, you cannot create IAM users with console access. For more information, see Invite team members.

An *IAM user* is an entity that you create in your AWS account. The IAM user represents the human user or workload who uses the IAM user to interact with AWS resources. An IAM user consists of a name and credentials.

An IAM user with administrator permissions is not the same thing as the AWS account root user. For more information about the root user, see AWS account root user.

How AWS identifies an IAM user

When you create an IAM user, IAM creates these ways to identify that user:

- A "friendly name" for the IAM user, which is the name that you specified when you created the IAM user, such as `Richard` or `Anaya`. These are the names you see in the AWS Management Console. Because IAM user names appear in Amazon Resource Names (ARNs), we do not recommend including personally identifying information in the IAM name. Refer to IAM name requirements for requirements and restrictions for IAM names.
- An Amazon Resource Name (ARN) for the IAM user. You use the ARN when you need to uniquely identify the IAM user across all of AWS. For example, you could use an ARN to specify the IAM user as a `Principal` in an IAM policy for an Amazon S3 bucket. An ARN for an IAM user might look like the following:

  `arn:aws:iam::`*`account-ID-without-hyphens`*`:user/Richard`
- A unique identifier for the IAM user. This ID is returned only when you use the API, Tools for Windows PowerShell, or AWS CLI to create the IAM user; you do not see this ID in the console.

For more information about these identifiers, see IAM identifiers.

IAM users and credentials

You can access AWS in different ways depending on the IAM user credentials:

- **Console password**: A password that the IAM user can type to sign in to interactive sessions such as the AWS Management Console. Disabling the password (console access) for an IAM user prevents them from signing in to the AWS Management Console using their sign-in credentials. It does not change their permissions or prevent them from accessing the console using an assumed role. IAM users with console access enabled can also use these same credentials to authenticate for AWS CLI and SDK access using the `aws login` AWS CLI command. These users will need to have SignInLocalDevelopmentAccess permissions. See Authentication and access credentials for the AWS CLI in the *AWS Command Line Interface User Guide* for more details.
- **Access keys**: Used to make programmatic calls to AWS. However, there are more secure alternatives to consider before you create access keys for IAM users. For more information, see Considerations and alternatives for long-term access keys in the *AWS General Reference*. If the IAM user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, AWS API, or the AWS Console Mobile Application.
- **SSH keys for use with CodeCommit**: An SSH public key in the OpenSSH format that can be used to authenticate with CodeCommit.
- **Server certificates**: SSL/TLS certificates that you can use to authenticate with some AWS services. We recommend that you use AWS Certificate Manager (ACM) to provision, manage, and deploy your server certificates. Use IAM only when you must support HTTPS connections in a region that is not supported by ACM. To learn which regions support ACM, see AWS Certificate Manager endpoints and quotas in the *AWS General Reference*.

You can choose the credentials that are right for your IAM user. When you use the AWS Management Console to create an IAM user, you must choose to at least include a console password or access keys. By default, a brand new IAM user created using the AWS CLI or AWS API has no credentials of any kind. You must create the type of credentials for an IAM user based on your use case.

You have the following options to administer passwords, access keys, and multi-factor authentication (MFA) devices:

- **Manage passwords for your IAM users.** Create and change the passwords that permit access to the AWS Management Console. Set a password policy to enforce a minimum password complexity. Allow IAM users to change their own passwords.
- **Manage access keys for your IAM users.** Create and update access keys for programmatic access to the resources in your account.
- **Enable multi-factor authentication (MFA) for the IAM user.** As a best practice, we recommend that you require multi-factor authentication for all IAM users in your account. With MFA, IAM users must provide two forms of identification: First, they provide the credentials that are part of their user identity (a password or access key). In addition, they provide a temporary numeric code that's generated on a hardware device or by an application on a smartphone or tablet.
- **Find unused passwords and access keys.** Anyone who has a password or access keys for your account or an IAM user in your account has access to your AWS resources. The security best practice is to remove passwords and access keys when IAM users no longer need them.
- **Download a credential report for your account.** You can generate and download a credential report that lists all IAM users in your account and the status of their various credentials, including passwords, access keys, and MFA devices. For passwords and access keys, the credential report shows how recently the password or access key has been used.

IAM users and permissions

By default, a new IAM user has no permissions to do anything. They are not authorized to perform any AWS operations or to access any AWS resources. An advantage of having individual IAM users is that you can assign permissions individually to each user. You might assign administrative permissions to a few users, who then can administer your AWS resources and can even create and manage other IAM users. In most cases, however, you want to limit a user's permissions to just the tasks (AWS actions or operations) and resources that are needed for the job.

Imagine a user named Diego. When you create the IAM user `Diego`, you create a password for him and attach permissions that let him launch a specific Amazon EC2 instance and read (`GET`) information from a table in an Amazon RDS database. For procedures on how to create IAM users and grant them initial credentials and permissions, see Create an IAM user in your AWS account. For procedures on how to change the permissions for existing users, see Change permissions for an IAM user. For procedures on how to change the user's password or access keys, see User passwords in AWS and Manage access keys for IAM users.

You can also add a permissions boundary to your IAM users. A permissions boundary is an advanced feature that allows you to use AWS managed policies to limit the maximum permissions that an identity-based policy can grant to an IAM user or role. For more information about policy types and uses, see Policies and permissions in AWS Identity and Access Management.

IAM users and accounts

Each IAM user is associated with one and only one AWS account. Because IAM users are defined within your AWS account, they don't need to have a payment method on file with AWS. Any AWS activity performed by IAM users in your account is billed to your account.

The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

IAM users as service accounts

An IAM user is a resource in IAM that has associated credentials and permissions. An IAM user can represent a person or an application that uses its credentials to make AWS requests. This is typically referred to as a *service account*. If you choose to use the long-term credentials of an IAM user in your application, **do not embed access keys directly into your application code.** The AWS SDKs and the AWS Command Line Interface allow you to put access keys in known locations so that you do not have to keep them in code. For more information, see Manage IAM User Access Keys Properly in the *AWS General Reference*. Alternatively, and as a best practice, you can use temporary security credentials (IAM roles) instead of long-term access keys.

After you create IAM users, you can View IAM users in your account and Rename an IAM user as needed.

How IAM users sign in to AWS

To sign in to the AWS Management Console as an IAM user, you must provide your account ID or account alias in addition to your user name and password. When your administrator created your IAM user in the console, they should have sent you your sign-in credentials, including your user name and the URL to your account sign-in page that includes your account ID or account alias.

```
https://My_AWS_Account_ID.signin.aws.amazon.com/console/
```

> **Tip**
>
> To create a bookmark for your account sign-in page in your web browser, you should manually type the sign-in URL for your account in the bookmark entry. Do not use your web browser bookmark feature because redirects can obscure the sign-in URL.

You can also sign in at the following general sign-in endpoint and type your account ID or account alias manually:

```
https://console.aws.amazon.com/
```

For convenience, the AWS sign-in page uses a browser cookie to remember the IAM user name and account information. The next time the user goes to any page in the AWS Management Console, the console uses the cookie to redirect the user to the account sign-in page.

You have access only to the AWS resources that your administrator specifies in the policy that is attached to your IAM user identity. To work in the console, you must have permissions to perform the actions that the console performs, such as listing and creating AWS resources. For more information, see Access management for AWS resources and Example IAM identity-based policies.

> **Note**
>
> If your organization has an existing identity system, you might want to create a single sign-on (SSO) option. SSO gives users access to the AWS Management Console for your account without requiring them to have an IAM user identity. SSO also eliminates the need for users to sign in to your organization's site and to AWS separately. For more information, see Enable custom identity broker access to the AWS console.

Logging sign-in details in CloudTrail

If you enable CloudTrail to log sign-in events to your logs, you need to be aware of how CloudTrail chooses where to log the events.

- If your users sign-in directly to a console, they are redirected to either a global or a regional sign-in endpoint, based on whether the selected service console supports regions. For example, the main console home page supports regions, so if you sign in to the following URL:

```
https://alias.signin.aws.amazon.com/console
```

you are redirected to a regional sign-in endpoint such as `https://us-east-2.signin.aws.amazon.com`, resulting in a regional CloudTrail log entry in the user's region's log:

On the other hand, the Amazon S3 console does not support regions, so if you sign in to the following URL

```
https://alias.signin.aws.amazon.com/console/s3
```

AWS redirects you to the global sign-in endpoint at `https://signin.aws.amazon.com`, resulting in a global CloudTrail log entry.

- You can manually request a certain regional sign-in endpoint by signing in to the region-enabled main console home page using a URL syntax like the following:

```
https://alias.signin.aws.amazon.com/console?region=ap-southeast-1
```

AWS redirects you to the `ap-southeast-1` regional sign-in endpoint and results in a regional CloudTrail log event.

For more information about CloudTrail and IAM, see Logging IAM events with CloudTrail.

If users need programmatic access to work with your account, you can create an access key pair (an access key ID and a secret access key) for each user. However, there are more secure alternatives to consider before you create access keys for users. For more information, see Considerations and alternatives for long-term access keys in the *AWS General Reference*.

Additional resources

The following resources can help you learn more about AWS sign-in.

- The AWS Sign-In User Guide helps you understand the different ways that you can sign in to Amazon Web Services (AWS), depending on what type of user you are.
- You can sign in up to five different identities simultaneously in a single web browser in the AWS Management Console. For details, see Signing in to multiple accounts in the *AWS Management Console Getting Started Guide*.

- To find the account ID or alias needed for sign-in, see Viewing your AWS account ID and Using an alias for your AWS account ID.

MFA enabled sign-in

Users who are configured with multi-factor authentication (MFA) devices must use their MFA devices to sign in to the AWS Management Console. After the user enters their sign-in credentials, AWS checks the user's account to see if MFA is required for that user.

> **Important**
>
> If you use access key and secret key credentials for direct AWS Management Console access with the AWS STS `GetFederationToken` API call, MFA will NOT be required. For more information, see Using access keys and secret key credentials for console access.

The following topics provide information on how users complete signing in when MFA is required.

Topics

- Multiple MFA devices enabled
- FIDO security key
- Virtual MFA device
- Hardware TOTP token

Multiple MFA devices enabled

If a user signs in to the AWS Management Console as an AWS account root user or IAM user with multiple MFA devices enabled for that account, they only need to use one MFA device to sign in. After the user authenticates with the user's password, they select which MFA device type they would like to use to finish authenticating. Then the user is prompted to authenticate with the type of device that they selected.

FIDO security key

If MFA is required for the user, a second sign-in page appears. The user needs to tap the FIDO security key.

> **Note**
>
> Google Chrome users should not choose any of the available options on the pop-up that asks to **Verify your identity with amazon.com**. You only need to tap on the security key.

Unlike other MFA devices, FIDO security keys do not go out of sync. Administrators can deactivate a FIDO security key if it's lost or broken. For more information, see Deactivating MFA devices (console).

For information on browsers that support WebAuthn and FIDO-compliant devices that AWS supports, see Supported configurations for using passkeys and security keys.

Virtual MFA device

If MFA is required for the user, a second sign-in page appears. In the **MFA code** box, the user must enter the numeric code provided by the MFA application.

If the MFA code is correct, the user can access the AWS Management Console. If the code is incorrect, the user can try again with another code.

A virtual MFA device can go out of sync. If a user cannot sign in to the AWS Management Console after several tries, the user is prompted to synchronize the virtual MFA device. The user can follow the on-screen prompts to synchronize the virtual MFA device. For information about how you can synchronize a device on behalf of a user in your AWS account, see Resynchronize virtual and hardware MFA devices.

Hardware TOTP token

If MFA is required for the user, a second sign-in page appears. In the **MFA code** box, the user must enter the numeric code provided by a hardware TOTP token.

If the MFA code is correct, the user can access the AWS Management Console. If the code is incorrect, the user can try again with another code.

A hardware TOTP token can go out of sync. If a user can't sign in to the AWS Management Console after several tries, the user is prompted to synchronize the MFA token device. The user can follow the on-screen prompts to synchronize the MFA token device. For information about how you can synchronize a device on behalf of a user in your AWS account, see Resynchronize virtual and hardware MFA devices.

Create an IAM user in your AWS account

> **Important**
>
> IAM best practices recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials instead of using IAM users with long-term credentials. We recommend that you only use IAM users for specific use cases not supported by federated users.
> If you're using our new AWS experience and want to provide human users with access to your projects or AWS Settings, you must add them to your team. In our new experience, you cannot create IAM users with console access. For more information, see Invite team members.

The process of creating an IAM user and enabling that user to perform tasks consists of the following steps:

1. Create the user in the AWS Management Console, the AWS CLI, Tools for Windows PowerShell, or using an AWS API operation. If you create the user in the AWS Management Console, then steps 1–4 are handled automatically, based on your choices. If you create the IAM users programmatically, then you must perform each of those steps individually.
2. Create credentials for the user, depending on the type of access the user requires:
   - **Enable console access – *optional***: If the user needs to access the AWS Management Console, create a password for the user. Disabling console access for a user prevents them from signing in to the AWS Management Console using their user name and password. It does not change their permissions or prevent them from accessing the console using an assumed role.

   > **Tip**
   >
   > Create only the credentials that the user needs. For example, for a user who requires access only through the AWS Management Console, do not create access keys.

3. Give the user permissions to perform the required tasks. We recommend that you put your IAM users in groups and manage permissions through policies that are attached to those groups. However, you can also grant permissions by attaching permissions policies directly to the user. If you use the console to add the user, you can copy the permissions from an existing user to the new user.

You can also add a permissions boundary to limit the user's permissions by specifying a policy that defines the maximum permissions that the user can have. Permissions boundaries don't grant any permissions.

For instructions on creating a custom permission policy to use to either grant permissions or set a permissions boundary, see Define custom IAM permissions with customer managed policies.

4. (Optional) Add metadata to the user by attaching tags. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
5. Provide the user with the necessary sign-in information. This includes the password and the console URL for the account sign-in page where the user provides those credentials. For more information, see How IAM users sign in to AWS.
6. (Optional) Configure multi-factor authentication (MFA) for the user. MFA requires the user to provide a one-time-use code each time he or she signs into the AWS Management Console.
7. (Optional) Give IAM users permissions to manage their own security credentials. (By default, IAM users do not have permissions to manage their own credentials.) For more information, see Permit IAM users to change their own passwords.

> **Note**
>
> If you use the console to create the user and you select **User must create a new password at next sign-in (recommended)**, the user has the required permissions.

For information about the permissions that you need in order to create a user, see Permissions required to access IAM resources.

For instructions on creating IAM users for specific use cases, see the following topics:

- Create an IAM user for emergency access
- Create an IAM user for workloads that can't use IAM roles

View IAM users

You can list the IAM users in your AWS account or in a specific IAM group, and list all the IAM groups that a user is in. For information about the permissions that you need in order to list users, see Permissions required to access IAM resources.

To list all the IAM users in your account

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.

The console displays the IAM users in your AWS account.

AWS CLI

Run the following command:

- `aws iam list-users`

API

Call the following operation:

- `ListUsers`

To list the IAM users in an IAM group

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **User groups**.
4. Choose the name of the user group.

The IAM users that are members of the group are listed on the **Users** tab.

AWS CLI

Run the following command:

- `aws iam get-group`

API

Call the following operation:

- `GetGroup`

To list all the IAM groups that a user is in

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user.
5. Select the **Groups** tab to display the list of groups that include the current user.

AWS CLI

Run the following command:

- `aws iam list-groups-for-user`

API

Call the following operation:

- `ListGroupsForUser`

Next steps

Once you have a list of your IAM users, you can rename, delete, or deactivate an IAM user using the following procedures.

- Rename an IAM user
- Remove or deactivate an IAM user

Rename an IAM user

> **Note**
>
> As a best practice, we recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials. If you follow the best practices, you are not managing IAM users and groups. Instead, your users and groups are managed outside of AWS and are able to access AWS resources as a *federated identity*. A federated identity is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. Federated identities use the groups defined by their identity provider. If you are using AWS IAM Identity Center, see Manage identities in IAM Identity Center in the *AWS IAM Identity Center User Guide* for information about creating users and groups in IAM Identity Center.

Amazon Web Services offers multiple tools for managing the IAM users in your AWS account. You can list the IAM users in your account or in a user group, or list all IAM groups that a user is a member of. You can rename or change the path of an IAM user. If you are moving to using federated identities instead of IAM users, you can delete an IAM user from your AWS account, or deactivate the user.

For more information about adding, changing, or removing managed policies for an IAM user, see Change permissions for an IAM user. For information about managing inline policies for IAM users, see Adding and removing IAM identity permissions, Edit IAM policies, and Delete IAM policies. As a best practice, use managed policies instead of inline policies. *AWS managed policies* grant permissions for many common use cases. Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they are available for use by all AWS customers. As a result, we recommend that you reduce permissions further by defining

customer managed policies that are specific to your use cases. For more information, see AWS managed policies. For more information about AWS managed policies that are designed for specific job functions, see AWS managed policies for job functions.

To learn about validating IAM policies, see IAM policy validation.

> **Tip**
>
> IAM Access Analyzer can analyze the services and actions that your IAM roles use, and then generate a fine-grained policy that you can use. After you test each generated policy, you can deploy the policy to your production environment. This ensures that you grant only the required permissions to your workloads. For more information about policy generation, see IAM Access Analyzer policy generation.

For information about managing IAM user passwords, see Manage passwords for IAM users.

Renaming an IAM user

To change a user's name or path, you must use the AWS CLI, Tools for Windows PowerShell, or AWS API. There is no option in the console to rename a user. For information about the permissions that you need in order to rename a user, see Permissions required to access IAM resources.

When you change a user's name or path, the following happens:

- Any policies attached to the user stay with the user under the new name.
- The user stays in the same IAM groups under the new name.
- The unique ID for the user remains the same. For more information about unique IDs, see Unique identifiers.
- Any resource or role policies that refer to the user *as a principal* (the user is being granted access) are automatically updated to use the new name or path. For example, any queue-based policies in Amazon SQS or resource-based policies in Amazon S3 are automatically updated to use the new name and path.

IAM does not automatically update policies that refer to the user *as a resource* to use the new name or path; you must manually do that. For example, imagine that user `Richard` has a policy attached to him that lets him manage his security credentials. If an administrator renames `Richard` to `Rich`, the administrator also needs to update that policy to change the resource from this:

```
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/Richard
```

to this:

```
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/Rich
```

This is true also if an administrator changes the path; the administrator needs to update the policy to reflect the new path for the user.

To rename a user

- AWS CLI: aws iam update-user
- AWS API: UpdateUser

Remove or deactivate an IAM user

Best practices recommend that you remove unused IAM users from your AWS account. If you want to retain the IAM users credentials for future use, instead of deleting them from the account you can deactivate the user's access. For more information, see Deactivating an IAM user.

> **Warning**
>
> Once an IAM user and its access keys are deleted, they cannot be restored or recovered.

Prerequisite – View IAM user access

Before you remove a user, review their recent service-level activity. This helps prevent removing access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Removing an IAM user (console)

When you use the AWS Management Console to remove an IAM user, IAM automatically deletes the following associated information:

- The IAM user identifier

- Any group memberships—that is, the IAM user is removed from any groups that the IAM user was a member of
- Any password associated with the IAM user
- All inline policies embedded in the IAM user (policies that were applied to the IAM user using user group permissions are not affected)

> **Note**
>
> IAM removes any managed policies attached to the IAM user when you delete the user, but does not delete managed policies.

- Any associated MFA device

To remove an IAM user (console)

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**, and then select the checkbox next to the IAM user name that you want to delete.
4. At the top of the page, choose **Delete**.

   > **Note**
   >
   > If any of the users have active access keys, you must deactivate the access keys before deleting the users. For more information, see To deactivate an access key for an IAM user.

5. In the confirmation dialog box, enter the username in the text input field to confirm the deletion of the user. Choose **Delete**.

The console displays a status notification that the IAM user was deleted.

Deleting an IAM user (AWS CLI)

Unlike the AWS Management Console, when you delete a IAM user with the AWS CLI, you must delete the items attached to the IAM user manually. This procedure illustrates the process.

To delete an IAM user from your AWS account (AWS CLI)

1. Delete the user's password, if the user has one.

   `aws iam delete-login-profile`

2. Delete the user's access keys, if the user has them.

   `aws iam list-access-keys` (to list the user's access keys) and `aws iam delete-access-key`

3. Delete the user's signing certificate. Note that when you delete a security credential, it's gone forever and can't be retrieved.

   `aws iam list-signing-certificates` (to list the user's signing certificates) and `aws iam delete-signing-certificate`

4. Delete the user's SSH public key, if the user has them.

   `aws iam list-ssh-public-keys` (to list the user's SSH public keys) and `aws iam delete-ssh-public-key`

5. Delete the user's Git credentials.

   `aws iam list-service-specific-credentials` (to list the user's git credentials) and `aws iam delete-service-specific-credential`

6. Deactivate the user's multi-factor authentication (MFA) device, if the user has one.

   `aws iam list-mfa-devices` (to list the user's MFA devices), `aws iam deactivate-mfa-device` (to deactivate the device), and `aws iam delete-virtual-mfa-device` (to permanently delete a virtual MFA device)

7. Delete the user's inline policies.

   `aws iam list-user-policies` (to list the inline policies for the user) and `aws iam delete-user-policy` (to delete the policy)

8. Detach any managed policies that are attached to the user.

`aws iam list-attached-user-policies` (to list the managed policies attached to the user) and `aws iam detach-user-policy` (to detach the policy)

9. Remove the user from any IAM groups.

   `aws iam list-groups-for-user` (to list the IAM groups to which the user belongs) and `aws iam remove-user-from-group`

10. Delete the user.

    `aws iam delete-user`

Deactivating an IAM user

You might need to deactivate an IAM user while they are temporarily away from your company. You can leave their IAM user credentials in place and still block their AWS access.

To deactivate a user, create and attach a policy to deny the user access to AWS. You can restore the user's access later.

Here are two examples of deny policies that you can attach to a user to deny their access.

The following policy does not include a time limit. You must remove the policy to restore the user's access.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*"
      }
   ]
}
```

The following policy includes a condition that starts the policy on December 24, 2024 at 11:59 PM (UTC) and ends it on February 28, 2025 at 11:59 PM (UTC).

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
          "DateGreaterThan": {"aws:CurrentTime": "2024-12-24T23:59:59Z"},
          "DateLessThan": {"aws:CurrentTime": "2025-02-28T23:59:59Z"}
          }
       }
   ]
}
```

Control IAM user access to the AWS Management Console

IAM users with permission who sign in to your AWS account through the AWS Management Console can access your AWS resources. The following list shows the ways that you can grant IAM users access to your AWS account resources through the AWS Management Console. It also shows how IAM users can access other AWS account features through the AWS website.

> **Note**
>
> There is no charge to use IAM.

The AWS Management Console

You create a password for each IAM user who needs access to the AWS Management Console. Users access the console through your IAM-enabled AWS account sign-in page. For information about accessing the sign-in page, see How to sign in to AWS in the *AWS Sign-In User Guide*. For information about creating passwords, see User passwords in AWS.

You can prevent an IAM user from accessing the AWS Management Console by removing their password. This prevents them from signing into the AWS Management Console using their sign-in credentials. It does not change their permissions or prevent them from accessing the console using an assumed role. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, AWS API, or the AWS Console Mobile Application.

Your AWS resources, such as Amazon EC2 instances, Amazon S3 buckets, and so on

Even if your IAM users have passwords, they still need permission to access your AWS resources. When you create an IAM user, that user has no permissions by default. To give your IAM users the permissions they need, you attach policies to them. If you have many IAM users who perform the same tasks with the same resources, you can assign those IAM users to a group. Then assign the permissions to that group. For information about creating IAM users and groups, see IAM Identities. For information about using policies to set permissions, see Access management for AWS resources.

AWS Discussion Forums

Anyone can read the posts on the AWS Discussion Forums. Users who want to post questions or comments to the AWS Discussion Forum can do so using their user name. The first time a user posts to the AWS Discussion Forum, the user is prompted to enter a nickname and email address. Only that user can use that nickname in the AWS Discussion Forums.

Your AWS account billing and usage information

You can grant users access your AWS account billing and usage information. For more information, see Controlling Access to Your Billing Information in the *AWS Billing User Guide*.

Your AWS account profile information

Users cannot access your AWS account profile information.

Your AWS account security credentials

Users cannot access your AWS account security credentials.

> **Note**
>
> IAM policies control access regardless of the interface. For example, you could provide a user with a password to access the AWS Management Console. The policies for that user (or any groups the user belongs to) would control what the user can do in the AWS

Management Console. Or, you could provide the user with AWS access keys for making API calls to AWS. The policies would control which actions the user could call through a library or client that uses those access keys for authentication.

Change permissions for an IAM user

You can change the permissions for an IAM user in your AWS account by changing its group memberships, by copying permissions from an existing user, by attaching policies directly to a user, or by setting a permissions boundary. A permissions boundary controls the maximum permissions that a user can have. Permissions boundaries are an advanced AWS feature.

For information about the permissions that you need in order to modify the permissions for a user, see Permissions required to access IAM resources.

Topics

- View user access
- Generate a policy based on a user's access activity
- Adding permissions to a user (console)
- Changing permissions for a user (console)
- To remove a permissions policy from a user (console)
- To remove the permissions boundary from a user (console)
- Adding and removing a user's permissions (AWS CLI or AWS API)

View user access

Before you change the permissions for a user, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Generate a policy based on a user's access activity

You might sometimes grant permissions to an IAM entity (user or role) beyond what they require. To help you refine the permissions that you grant, you can generate an IAM policy that is based on the access activity for an entity. IAM Access Analyzer reviews your AWS CloudTrail logs and

generates a policy template that contains the permissions that have been used by the entity in your specified date range. You can use the template to create a managed policy with fine-grained permissions and then attach it to the IAM entity. That way, you grant only the permissions that the user or role needs to interact with AWS resources for your specific use case. To learn more, see IAM Access Analyzer policy generation.

Adding permissions to a user (console)

IAM offers three ways to add permissions policies to a user:

- **Add the IAM user to an IAM group** – Make the user a member of a group. The policies from the group are attached to the user.
- **Copy permissions from an existing IAM user** – Copy all group memberships, attached managed policies, inline policies, and any existing permissions boundaries from the source user.
- **Attach policies directly to the IAM user** – Attach a managed policy directly to the user. For easier permissions management, attach your policies to a group and then make IAM users members of the appropriate groups.

> **Important**
>
> If the user has a permissions boundary, then you cannot add more permissions to the user than are allowed by the permissions boundary.

To add permissions by adding the IAM user to a group

Adding an IAM user to an IAM group updates the user's permissions with the permissions defined for the group immediately.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user.

5. Select the **Groups** tab to display the list of groups that include the current user.
6. Choose **Add user to groups**.
7. Select the checkbox for each group that you want the user to join. The list shows each group's name and the policies that the user receives if made a member of that group.
8. (Optional) You can choose **Create group** to define a new group. This is useful if you want to add the user to a group with different attached policies than the existing groups:

   a. In the new tab, for **User group name**, type a name for your new group.

   > **Note**
   >
   > The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas. Group names can be a combination of up to 128 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), and hyphen (-). Names must be unique within an account. They are not distinguished by case. For example, you cannot create two groups named *TESTGROUP* and *testgroup*.

   b. Select one or more checkboxes for the managed policies that you want to attach to the group. You can also create a new managed policy by choosing **Create policy**. If you do, return to this browser tab or window when the new policy is done; choose **Refresh**; and then choose the new policy to attach it to your group. For more information, see Define custom IAM permissions with customer managed policies.

   c. Choose **Create user group**.

   d. Return to the original tab, refresh your list of groups. Then select the checkbox for your new group.
9. Choose **Add user to group(s)**.

The console displays a status message informing you that the user has been added to the groups you specified.

To add permissions by copying from another IAM user

If you choose to add permissions to an IAM user by copying permissions, IAM copies all group memberships, attached managed policies, inline policies, and any existing permissions boundaries from the specified user and applies them immediately to the currently selected user.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user.
5. On the **Permissions** tab choose and select **Add permissions**.
6. On the **Add permissions** page, choose **Copy permissions**. The list displays available IAM users along with their group memberships and attached policies.
7. Select the radio button next to the user whose permissions you want to copy.
8. Choose **Next** to see the list of changes that are to be made to the user. Then choose **Add permissions**.

The console displays a status message informing you that the permissions were copied from the IAM user you specified.

To add permissions by attaching policies directly to the IAM user

You can attach a managed policy directly to an IAM user. The updated permissions are applied immediately.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user.
5. On the **Permissions** tab, choose and select **Add permissions**.
6. On the **Add permissions** page, choose **Attach policies directly**. The **Permissions policies** list displays available policies along with their policy types and attached entities.

7. Select the radio button next to the **Policy name** you want to attach.
8. Choose **Next** to see the list of changes that are to be made to the user. Then choose **Add permissions**.

The console displays a status message informing you that the policy was added to the IAM user you specified.

To set the permissions boundary for an IAM user

A permissions boundary is an advanced feature for managing permissions in AWS that is used to set the maximum permissions that an IAM user can have. Setting a permissions boundary immediately restricts the IAM user permissions to the boundary, regardless of the other permissions granted.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user whose permissions boundary you want to change.
5. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section and then choose **Set permissions boundary**.
6. On the **Set permissions boundary** page, under **Permissions policies** select the policy that you want to use for the permissions boundary.
7. Choose **Set boundary**.

The console displays a status message informing you that the permissions boundary has been added.

Changing permissions for a user (console)

IAM allows you to change the permissions that are associated with a user in the following ways:

- **Edit a permissions policy** – Edit a user's inline policy, the inline policy of the user's group, or edit a managed policy that is attached to the user directly or from a group. If the user has a permissions boundary, then you cannot provide more permissions than are allowed by the policy that was used as the user's permissions boundary.
- **Changing the permissions boundary** – Change the policy that is used as the permissions boundary for the user. This can expand or restrict the maximum permissions that a user can have.

Editing a permissions policy attached to a user

Changing permissions updates the user's access immediately.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user whose permissions boundary you want to change.
5. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section.
6. Choose the name of the policy that you want to edit to view details about the policy. Choose the **Entities attached** tab to view other entities (IAM users, groups, and roles) that might be affected if you edit the policy.
7. Choose the **Permissions** tab and review the permissions granted by the policy. To make changes to the permissions, choose **Edit**.
8. Edit the policy and resolve any policy validation recommendations. For more information, see Edit IAM policies.
9. Choose **Next**, review the policy summary, and then choose **Save changes**.

The console displays a status message informing you that the policy has been updated.

To change the permissions boundary for a user

Changing a permissions boundary updates the user's access immediately.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user whose permissions boundary you want to change.
5. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section and then choose **Change boundary**.
6. Select the policy that you want to use for the permissions boundary.
7. Choose **Set boundary**.

The console displays a status message informing you that the permissions boundary has been changed.

To remove a permissions policy from a user (console)

Removing a permissions policy updates the user's access immediately.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. Choose the name of the user whose permissions policies you want to remove.
5. Choose the **Permissions** tab.

6. If you want to remove permissions by removing an existing policy, view the **Attached via** column to understand how the user is getting that policy before choosing **Remove** to remove the policy:

   - If the policy applies because of group membership, then choosing **Remove** removes the user from the group. Remember that you might have multiple policies attached to a single group. If you remove a user from a group, the user loses access to *all* policies that it received through that group membership.
   - If the policy is a managed policy attached directly to the user, then choosing **Remove** detaches the policy from the user. This does not affect the policy itself or any other entity that the policy might be attached to.
   - If the policy is an inline embedded policy, then choosing **Remove** removes the policy from IAM. Inline policies that are attached directly to a user exist only on that user.

   If the policy was granted to the user through a group membership, the console displays a status message informing you that the IAM user was removed from the IAM group. If the policy directly attached or inline, the status message informs you that the policy has been removed.

To remove the permissions boundary from a user (console)

Removing the permissions boundary updates the user's access immediately.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. In the **Users** list, choose the name of the IAM user whose permissions boundary you want to remove.
5. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section.
6. Choose **Change boundary**. To confirm that you want to remove the permissions boundary, in the confirmation dialog, choose **Remove boundary**.

The console displays a status message informing you that the permissions boundary has been removed.

Adding and removing a user's permissions (AWS CLI or AWS API)

To add or remove permissions programmatically, you must add or remove the group memberships, attach or detach the managed policies, or add or delete the inline policies. For more information, see the following topics:

- Edit users in IAM groups
- Adding and removing IAM identity permissions

User passwords in AWS

You can manage passwords for IAM users in your account. IAM users need passwords in order to access the AWS Management Console. Users do not need passwords to access AWS resources programmatically by using the AWS CLI, Tools for Windows PowerShell, the AWS SDKs or APIs. For those environments, you have the option of assigning IAM users access keys. However, there are other more secure alternatives to access keys that we recommend you consider first. For more information, see AWS security credentials.

> **Note**
>
> If one of your IAM users loses or forgets their password, you *cannot* retrieve it from IAM. Depending on your settings, either the user or the administrator must create a new password.

Contents

- Set an account password policy for IAM users
- Manage passwords for IAM users
- Permit IAM users to change their own passwords
- How an IAM user changes their own password

Set an account password policy for IAM users

You can set a custom password policy on your AWS account to specify complexity requirements and mandatory rotation periods for your IAM users' passwords. If you don't set a custom password policy, IAM user passwords must meet the default AWS password policy. For more information, see Custom password policy options.

Topics

- Rules for setting a password policy
- Permissions required to set a password policy
- Default password policy
- Custom password policy options
- To set a password policy (console)
- To change a password policy (console)
- To delete a custom password policy (console)
- Setting a password policy (AWS CLI)
- Setting a password policy (AWS API)

Rules for setting a password policy

The IAM password policy does not apply to the AWS account root user password or IAM user access keys. If a password expires, the IAM user can't sign in to the AWS Management Console but can continue to use their access keys.

When you create or change a password policy, most of the password policy settings are enforced the next time your users change their passwords. However, some of the settings are enforced immediately. For example:

- When the minimum length and character type requirements change, these settings are enforced the next time that your users change their passwords. Users are not forced to change their existing passwords, even if the existing passwords do not adhere to the updated password policy.
- When you set a password expiration period, the expiration period is enforced immediately. For example, assume that you set a password expiration period of 90 days. In that case, the password expires for all IAM users whose existing password is older than 90 days. Those users are required to change their password the next time that they sign in.

You can't create a "lockout policy" to lock a user out of the account after a specified number of failed sign-in attempts. For enhanced security, we recommend that you combine a strong password policy with multi-factor authentication (MFA). For more information about MFA, see AWS Multi-factor authentication in IAM.

Permissions required to set a password policy

You must configure permissions to allow an IAM entity (user or role) to view or edit their account password policy. You can include the following password policy actions in an IAM policy:

- `iam:GetAccountPasswordPolicy` – Allows the entity to view the password policy for their account
- `iam:DeleteAccountPasswordPolicy` – Allows the entity to delete the custom password policy for their account and revert to the default password policy
- `iam:UpdateAccountPasswordPolicy` – Allows the entity to create or change the custom password policy for their account

The following policy allows full access to view and edit the account password policy. To learn how to create an IAM policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessPasswordPolicy",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccountPasswordPolicy",
                "iam:DeleteAccountPasswordPolicy",
                "iam:UpdateAccountPasswordPolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

For information about the permissions required for an IAM user to change their own password, see Permit IAM users to change their own passwords.

Default password policy

If an administrator does not set a custom password policy, IAM user passwords must meet the default AWS password policy.

The default password policy enforces the following conditions:

- Minimum password length of 8 characters and a maximum length of 128 characters
- Minimum of three of the following mix of character types: uppercase, lowercase, numbers, and non-alphanumeric character (`! @ # $ % ^ & * ( ) _ + - = [ ] { } | '`)
- Not be identical to your AWS account name or email address
- Never expire password

Custom password policy options

When you configure a custom password policy for your account, you can specify the following conditions:

- **Password minimum length** – You can specify a minimum of 6 characters and a maximum of 128 characters.
- **Password strength** – You can select any of the following checkboxes to define the strength of your IAM user passwords:
  - Require at least one uppercase letter from the Latin alphabet (A–Z)
  - Require at least one lowercase letter from the Latin alphabet (a–z)
  - Require at least one number
  - Require at least one nonalphanumeric character `! @ # $ % ^ & * ( ) _ + - = [ ] { } | '`
- **Turn on password expiration** – You can select and specify a minimum of 1 and a maximum of 1,095 days that IAM user passwords are valid after they are set. For example, if you specify an expiration of 90 days, it immediately impacts all of your users. For users with passwords older than 90 days, when they log into the console after the change, they must set a new password. Users with passwords 75-89 days old receive an AWS Management Console warning about their password expiration. IAM users can change their password at any time if they have permission.

When they set a new password, the expiration period for that password starts over. An IAM user can have only one valid password at a time.

- **Password expiration requires administrator reset** – Select this option to prevent IAM users from using the AWS Management Console to update their own passwords after the password expires. Before you select this option, confirm that your AWS account has more than one user with administrative permissions to reset IAM user passwords. Administrators with `iam:UpdateLoginProfile` permission can reset IAM user passwords. IAM users with `iam:ChangePassword` permission and active access keys can reset their own IAM user console password programmatically. If you clear this checkbox, IAM users with expired passwords must still set a new password before they can access the AWS Management Console.
- **Allow users to change their own password** – You can permit all IAM users in your account to change their own password. This gives users access to the `iam:ChangePassword` action for only their user and to the `iam:GetAccountPasswordPolicy` action. This option does not attach a permissions policy to each user. Rather, IAM applies the permissions at the account-level for all users. Alternatively, you can allow only some users to manage their own passwords. To do so, you clear this checkbox. For more information about using policies to limit who can manage passwords, see Permit IAM users to change their own passwords.
- **Prevent password reuse** – You can prevent IAM users from reusing a specified number of previous passwords. You can specify a minimum number of 1 and a maximum number of 24 previous passwords that can't be repeated.

To set a password policy (console)

You can use the AWS Management Console to create, change, or delete a custom password policy. Changes to the password policy apply to new IAM users created after this policy change and existing IAM users when they change their passwords.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Account settings**.
4. In the **Password policy** section, choose **Edit**.
5. Choose **Custom** to use a custom password policy.

6. Select the options that you want to apply to your password policy and choose **Save changes**.
7. Confirm that you want to set a custom password policy by choosing **Set custom**.

The console displays a status message informing you that password requirements for IAM users have been updated.

To change a password policy (console)

You can use the AWS Management Console to create, change, or delete a custom password policy. Changes to the password policy apply to new IAM users created after this policy change and existing IAM users when they change their passwords.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Account settings**.
4. In the **Password policy** section, choose **Edit**.
5. Select the options that you want to apply to your password policy and choose **Save changes**.
6. Confirm that you want to set a custom password policy by choosing **Set custom**.

The console displays a status message informing you that password requirements for IAM users have been updated.

To delete a custom password policy (console)

You can use the AWS Management Console to create, change, or delete a custom password policy. Changes to the password policy apply to new IAM users created after this policy change and existing IAM users when they change their passwords.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Account settings**.
4. In the **Password policy** section, choose **Edit**.
5. Choose **IAM default** to delete the custom password policy and choose **Save changes**.
6. Confirm that you want to set the IAM default password policy by choosing **Set default**.

   The console displays a status message informing you that the password policy is set to IAM default.

Setting a password policy (AWS CLI)

You can use the AWS Command Line Interface to set a password policy.

To manage the custom account password policy from the AWS CLI

Run the following commands:

- To create or change the custom password policy: `aws iam update-account-password-policy`
- To view the password policy: `aws iam get-account-password-policy`
- To delete the custom password policy: `aws iam delete-account-password-policy`

Setting a password policy (AWS API)

You can use AWS API operations to set a password policy.

To manage the custom account password policy from the AWS API

Call the following operations:

- To create or change the custom password policy: `UpdateAccountPasswordPolicy`

- To view the password policy: `GetAccountPasswordPolicy`
- To delete the custom password policy: `DeleteAccountPasswordPolicy`

Manage passwords for IAM users

IAM users who use the AWS Management Console to work with AWS resources must have a password in order to sign in. You can create, change, or delete a password for an IAM user in your AWS account.

After you have assigned a password to a user, the user can sign in to the AWS Management Console using the sign-in URL for your account, which looks like this:

```
https://12-digit-AWS-account-ID or alias.signin.aws.amazon.com/console
```

For more information about how IAM users sign in to the AWS Management Console, see How to sign in to AWS in the *AWS Sign-In User Guide*.

Even if your users have their own passwords, they still need permissions to access your AWS resources. By default, a user has no permissions. To give your users the permissions they need, you assign policies to them or to the groups they belong to. For information about creating users and groups, see IAM Identities. For information about using policies to set permissions, see Change permissions for an IAM user.

You can grant users permission to change their own passwords. For more information, see Permit IAM users to change their own passwords. For information about how users access your account sign-in page, see How to sign in to AWS in the *AWS Sign-In User Guide*.

Topics

- Creating, changing, or deleting an IAM user password (console)

Creating, changing, or deleting an IAM user password (console)

You can use the AWS Management Console to manage passwords for your IAM users.

The access needs of your users can change over time. You might need to enable a user intended for CLI access to have console access, change a user's password because they receive the email with their credentials, or delete a user when they leave your organization or no longer need AWS access.

To create an IAM user password (console)

Use this procedure to give a user console access by creating a password that is associated with the username.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. Choose the name of the user whose password you want to create.
5. Choose the **Security credentials** tab, and then under **Console sign-in**, choose **Enable console access**.
6. In the **Enable console access** dialog box, select **Reset password**, then choose whether to have IAM generate a password or create a custom password:
   - To have IAM generate a password, choose **Autogenerated password**.
   - To create a custom password, choose **Custom password**, and type the password.

   > **Note**
   >
   > The password that you create must meet the account's password policy.
7. To require the user to create a new password when signing in, choose **Require password change at the next sign-in**.
8. To require the user to use the new password immediately, select **Revoke active console sessions**. This attaches an inline policy to the IAM user that denies the user access to resources if their credentials are older than the time specified by the policy.
9. Choose **Reset password**
10. The **Console password** dialog informs you that you have enabled the user's new password. To view the password so you can share it with the user, choose **Show** in the **Console password** dialog box. Select **Download .csv file** to download a file with the user's credentials.

> **Important**
>
> For security reasons, you cannot access the password after completing this step, but you can create a new password at any time.

The console displays a status message informing you that console access has been enabled.

To change the password for an IAM user (console)

Use this procedure to update a password that is associated with the username.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.
2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. Choose the name of the user whose password you want to change.
5. Choose the **Security credentials** tab, and then under **Console sign-in**, choose **Manage console access**.
6. In the **Manage console access** dialog box, select **Reset password**, then choose whether to have IAM generate a password or create a custom password:
   - To have IAM generate a password, choose **Autogenerated password**.
   - To create a custom password, choose **Custom password**, and type the password.

   > **Note**
   >
   > The password that you create must meet the account's password policy.
7. To require the user to create a new password when signing in, choose **Require password change at the next sign-in**.

8. To require the user to use the new password immediately, select **Revoke active console sessions**. This attaches an inline policy to the IAM user that denies the user access to resources if their credentials are older than the time specified by the policy.
9. Choose **Reset password**
10. The **Console password** dialog informs you that you have enabled the user's new password. To view the password so you can share it with the user, choose **Show** in the **Console password** dialog box. Select **Download .csv file** to download a file with the user's credentials.

    > **Important**
    >
    > For security reasons, you cannot access the password after completing this step, but you can create a new password at any time.

The console displays a status message informing you that console access has been updated.

To delete (disable) an IAM user password (console)

Use this procedure to delete a password that is associated with the username, removing console access for the user.

> **Important**
>
> You can prevent an IAM user from accessing the AWS Management Console by removing their password. This prevents them from signing in to the AWS Management Console using their sign-in credentials. It does not change their permissions or prevent them from accessing the console using an assumed role. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, AWS API, or the AWS Console Mobile Application.

Console

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide*.

2. On the **IAM Console Home** page, in the left navigation pane, enter your query in the **Search IAM** text box.
3. In the navigation pane, choose **Users**.
4. Choose the name of the user whose password you want to delete.
5. Choose the **Security credentials** tab, and then under **Console sign-in**, choose **Manage console access**.
6. To require the user to stop using the console immediately, select **Revoke active console sessions**. This attaches an inline policy to the IAM user that denies the user access to resources if their credentials are older than the time specified by the policy.
7. Choose **Disable access**

The console displays a status message informing you that console access has been disabled.

Creating, changing, or deleting an IAM user password (AWS CLI)

You can use the AWS CLI to manage passwords for your IAM users.

To create a password (AWS CLI)

1. (Optional) To determine whether a user has a password, run this command: aws iam get-login-profile
2. To create a password, run this command: aws iam create-login-profile

To change a user's password (AWS CLI)

1. (Optional) To determine whether a user has a password, run this command: aws iam get-login-profile
2. To change a password, run this command: aws iam update-login-profile

To delete (disable) a user's password (AWS CLI)

1. (Optional) To determine whether a user has a password, run this command: aws iam get-login-profile
2. (Optional) To determine when a password was last used, run this command: aws iam get-user
3. To delete a password, run this command: aws iam delete-login-profile

> **Important**
>
> When you delete a user's password, the user can no longer sign in to the AWS Management Console. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, or AWS API function calls. When you use the AWS CLI, Tools for Windows PowerShell, or AWS API to delete a user from your AWS account, you must first delete the password using this operation. For more information, see Deleting an IAM user (AWS CLI).

To revoke a user's active console sessions before a specified time (AWS CLI)

1. To embed an inline policy that revokes an IAM user's active console sessions before a specified time, use the following inline policy and run this command: aws iam put-user-policy

   This inline policy denies all permissions and includes the `aws:TokenIssueTime` condition key. It revokes the user's active console sessions before the specified time in the `Condition` element of the inline policy. Replace the `aws:TokenIssueTime` condition key value with your own value.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": {
       "Effect": "Deny",
       "Action": "*",
       "Resource": "*",
       "Condition": {
         "DateLessThan": {
           "aws:TokenIssueTime": "2014-05-07T23:47:00Z"
         }
       }
     }
   }
   ```

2. (Optional) To list the names of the inline policies embedded in the IAM user, run this command: aws iam list-user-policies

3. (Optional) To view the named inline policy embedded in the IAM user, run this command: aws iam get-user-policy

Creating, changing, or deleting an IAM user password (AWS API)

You can use the AWS API to manage passwords for your IAM users.

To create a password (AWS API)

1. (Optional) To determine whether a user has a password, call this operation: GetLoginProfile
2. To create a password, call this operation: CreateLoginProfile

To change a user's password (AWS API)

1. (Optional) To determine whether a user has a password, call this operation: GetLoginProfile
2. To change a password, call this operation: UpdateLoginProfile

To delete (disable) a user's password (AWS API)

1. (Optional) To determine whether a user has a password, run this command: GetLoginProfile
2. (Optional) To determine when a password was last used, run this command: GetUser
3. To delete a password, run this command: DeleteLoginProfile

> **Important**
>
> When you delete a user's password, the user can no longer sign in to the AWS Management Console. If the user has active access keys, they continue to function and allow access through the AWS CLI, Tools for Windows PowerShell, or AWS API function calls. When you use the AWS CLI, Tools for Windows PowerShell, or AWS API to delete a user from your AWS account, you must first delete the password using this operation. For more information, see Deleting an IAM user (AWS CLI).

To revoke a user's active console sessions before a specified time (AWS API)

1. To embed an inline policy that revokes an IAM user's active console sessions before a specified time, use the following inline policy and run this command: PutUserPolicy

This inline policy denies all permissions and includes the `aws:TokenIssueTime` condition key. It revokes the user's active console sessions before the specified time in the `Condition` element of the inline policy. Replace the `aws:TokenIssueTime` condition key value with your own value.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
      "DateLessThan": {
        "aws:TokenIssueTime": "2014-05-07T23:47:00Z"
      }
    }
  }
}
```

2. (Optional) To list the names of the inline policies embedded in the IAM user, run this command: ListUserPolicies
3. (Optional) To view the named inline policy embedded in the IAM user, run this command: GetUserPolicy

Permit IAM users to change their own passwords

> **Note**
>
> Users with federated identities will use the process defined by their identity provider to change their passwords. As a best practice, require human users to use federation with an identity provider to access AWS using temporary credentials.

You can grant IAM users the permission to change their own passwords for signing in to the AWS Management Console. You can do this in one of two ways:

- Allow all IAM users in the account to change their own passwords.
- Allow only selected IAM users to change their own passwords. In this scenario, you disable the option for all users to change their own passwords and you use an IAM policy to grant permissions to only some users. This approach allows those users to change their own passwords and optionally other credentials like their own access keys.

> **Important**
>
> We recommend that you set a custom password policy that requires IAM users to create strong passwords.

To allow all IAM users to change their own passwords

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Account settings**.
3. In the **Password policy** section, choose **Edit**.
4. Choose **Custom** to use a custom password policy.
5. Select **Allow users to change their own password**, and then choose **Save changes**. This allows all users in the account access to the `iam:ChangePassword` action for only their user and to the `iam:GetAccountPasswordPolicy` action.
6. Provide users with the following instructions for changing their passwords: How an IAM user changes their own password.

AWS CLI

Run the following command:

- `aws iam update-account-password-policy`

API

To update the account password policy, call the following operation:

- UpdateAccountPasswordPolicy

To allow selected IAM users to change their own passwords

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Account settings**.
3. In the **Password policy** section, make sure that **Allow users to change their own password** is not selected. If this checkbox is selected, all users can change their own passwords. (See the previous procedure.)
4. Create the users who should be allowed to change their own password, if they do not already exist. For details, see Create an IAM user in your AWS account.
5. (Optional) Create an IAM group for the users who should be allowed to change their passwords, and then add the users from the previous step to the group. For details, see IAM user groups.
6. Assign the following policy to the group. For more information, see Manage IAM policies.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:GetAccountPasswordPolicy",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:ChangePassword",
      "Resource": "arn:aws:iam::*:user/${aws:username}"
    }
  ]
}
```

This policy grants access to the ChangePassword action, which lets users change only their own passwords from the console, the AWS CLI, Tools for Windows PowerShell, or the API. It also grants access to the GetAccountPasswordPolicy action, which lets the user view the current password policy; this permission is required so that the user can view the account password policy on the **Change password** page. The user must be allowed to read the current password policy to make sure that the changed password meets the requirements of the policy.

7. Provide users with the following instructions for changing their passwords: How an IAM user changes their own password.

For more information

For more information on managing credentials, see the following topics:

- Permit IAM users to change their own passwords
- User passwords in AWS
- Set an account password policy for IAM users
- Manage IAM policies
- How an IAM user changes their own password

How an IAM user changes their own password

If you have been granted permission to change your own IAM user password, you can use a special page in the AWS Management Console to do this. You can also use the AWS CLI or AWS API.

Topics

- Permissions required
- How IAM users change their own password (console)
- How IAM users change their own password (AWS CLI or AWS API)

Permissions required

To change the password for your own IAM user, you must have the permissions from the following policy: AWS: Allows IAM users to change their own console password on the Security credentials page.

How IAM users change their own password (console)

The following procedure describes how IAM users can use the AWS Management Console to change their own password.

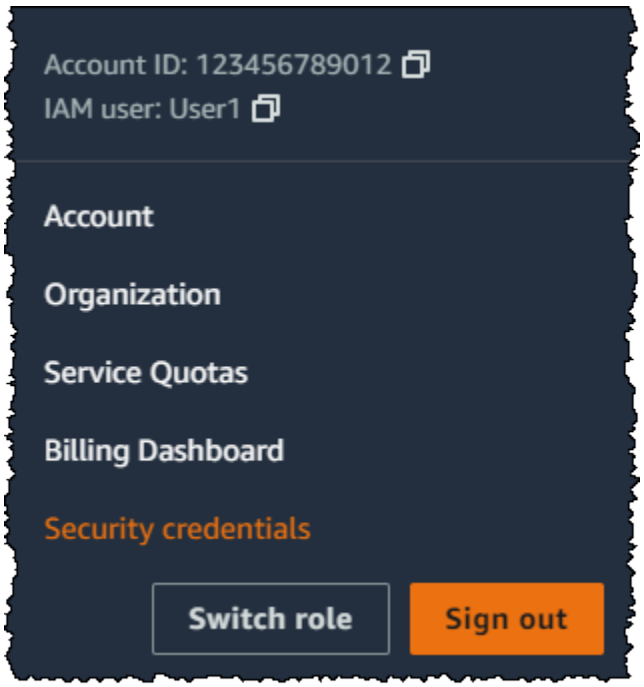
To change your own IAM user password (console)

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

   > **Note**
   >
   > For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

   To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. On the **AWS IAM credentials** tab, choose **Update password**.
4. For **Current password**, enter your current password. Enter a new password for **New password** and **Confirm new password**. Then choose **Update password**.

> **Note**
>
> The new password must meet the requirements of the account password policy. For more information, see Set an account password policy for IAM users.

How IAM users change their own password (AWS CLI or AWS API)

The following procedure describes how IAM users can use the AWS CLI or AWS API to change their own password.

To change your own IAM password, use the following:

- AWS CLI: `aws iam change-password`
- AWS API: `ChangePassword`

Manage access keys for IAM users

Important

As a best practice, use temporary security credentials (such as IAM roles) instead of creating long-term credentials like access keys. Before creating access keys, review the alternatives to long-term access keys.

Access keys are long-term credentials for an IAM user or the AWS account root user. You can use access keys to sign programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK). For more information, see Programmatic access with AWS security credentials.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). You must use both the access key ID and secret access key together to authenticate your requests.

When you create an access key pair, save the access key ID and secret access key in a secure location. The secret access key can be retrieved only at the time you create it. If you lose your secret access key, you must delete the access key and create a new one. For more instructions, see Update access keys.

You can have a maximum of two access keys per user.

Important

IAM users with access keys are an account security risk. Manage your access keys securely. Do not provide your access keys to unauthorized parties, even to help find your account identifiers. By doing this, you might give someone permanent access to your account. When working with access keys, be aware of the following:

- **Do NOT** use your account's root credentials to create access keys.
- **Do NOT** put access keys or credential information in your application files.
- **Do NOT** include files that contain access keys or credential information in your project area.
- Access keys or credential information stored in the shared AWS credentials file are stored in plaintext.

Monitoring recommendations

After creating access keys:

- Use AWS CloudTrail to monitor access key usage and detect any unauthorized access attempts. For more information, see Logging IAM and AWS STS API calls with AWS CloudTrail.
- Set up CloudWatch alarms to notify administrators for denied access attempts to help detect malicious activities. For more information, see the Amazon CloudWatch User Guide.
- Regularly review, update, and delete access keys as needed.

The following topics detail management tasks associated with access keys.

Topics

- Control the use of access keys by attaching an inline policy to an IAM user
- Permissions required to manage access keys
- How IAM users can manage their own access keys
- How an IAM administrator can manage IAM user access keys
- Update access keys
- Secure access keys

Control the use of access keys by attaching an inline policy to an IAM user

As a best practice we recommend that workloads use temporary credentials with IAM roles to access AWS. IAM users with access keys should be assigned least privilege access and have multi-factor authentication (MFA) enabled. For more information about assuming IAM roles, see Methods to assume a role.

However, if you're creating a proof of concept test of a service automation or other short-term use case, and you choose to run workloads using an IAM user with access keys we recommend that you use policy conditions to further restrict access of their IAM user credentials.

In this situation you can either create a time-bound policy that expires the credentials after the specified time or, if you are running a workload from a secure network, you can use an IP restriction policy.

For both these use cases, you can use an inline policy that's attached to the IAM user that has access keys.

To configure a time-bound policy for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and then select the user for the short-term use case. If you haven't created the user yet, you can create the user now.
3. On the user **Details** page, choose the **Permissions** tab.
4. Choose **Add permissions** and then select **Create inline policy**.
5. In the **Policy editor** section, select **JSON** to display the JSON editor.
6. In the JSON editor, enter the following policy, replacing the value for the `aws:CurrentTime` timestamp with your desired expiration date and time:

   JSON

   ```
   {
   "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Deny",
         "Action": "*",
         "Resource": "*",
         "Condition": {
         "DateGreaterThan": {
         "aws:CurrentTime": "2025-03-01T00:12:00Z"
           }
         }
       }
     ]
   }
   ```

   This policy uses the `Deny` effect to restrict all actions on all resources after the specified date. The `DateGreaterThan` condition compares the current time with the timestamp you set.
7. Select **Next** to proceed to the **Review and create** page. In **Policy** details, under **Policy name** enter a name for the policy and then choose **Create policy**.

After the policy is created, it's displayed on the **Permissions** tab for the user. When the current time is greater than or equal to the time specified in the policy, the user will no longer have access to

AWS resources. Make sure to inform workload developers of the expiration date you specified for these access keys.

To configure an IP restriction policy for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and then select the user that will run the workload from the secure network. If you haven't created the user yet, you can create the user now.
3. On the user **Details** page, choose the **Permissions** tab.
4. Choose **Add permissions** and then select **Create inline policy**.
5. In the **Policy editor** section, select **JSON** to display the JSON editor.
6. Copy the following IAM policy into the JSON editor, and change the public IPv4 or IPv6 addresses, or ranges to your needs. You can use https://checkip.amazonaws.com/ to determine your current public IP address. You can specify individual IP addresses, or ranges of IP addresses using slash notation. For more information, see aws:SourceIp.

> **Note**
>
> The IP addresses must not be obfuscated by a VPN or a proxy server.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid":"IpRestrictionIAMPolicyForIAMUser",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "NotIpAddress": {
          "aws:SourceIp": [
            "203.0.113.0/24",
            "2001:DB8:1234:5678::/64",
            "203.0.114.1"
```

```
          ]
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

This policy example denies the use of an IAM user's access keys with this policy applied, unless the request originated from the networks (specified in CIDR notation) "203.0.113.0/24", "2001:DB8:1234:5678::/64", or the specific IP address "203.0.114.1"

7. Select **Next** to proceed to the **Review and create** page. In **Policy** details, under **Policy name** enter a name for the policy and then choose **Create policy**.

After the policy is created, it's displayed on the **Permissions** tab for the user.

You could also apply this policy as a service control policy (SCP) across multiple AWS accounts in AWS Organizations, we recommend using an additional condition, `aws:PrincipalArn` to make this policy statement only apply to IAM users within the AWS accounts subject to this SCP. The following policy includes that update:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IpRestrictionServiceControlPolicyForIAMUsers",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "NotIpAddress": {
          "aws:SourceIp": [
            "203.0.113.0/24",
            "2001:DB8:1234:5678::/64",
```

```
            "203.0.114.1"
          ]
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        },
        "ArnLike": {
          "aws:PrincipalArn": "arn:aws:iam::*:user/*"
        }
      }
    }
  ]
}
```

Permissions required to manage access keys

> **Note**
>
> `iam:TagUser` is an optional permission for adding and editing descriptions for the access key. For more information, see Tag IAM users

To create access keys for your own IAM user, you must have the permissions from the following policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:CreateAccessKey",
                "iam:GetUser",
                "iam:ListAccessKeys",
                "iam:TagUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
```

```
        }
    ]
}
```

To update access keys for your own IAM user, you must have the permissions from the following policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:CreateAccessKey",
                "iam:DeleteAccessKey",
                "iam:GetAccessKeyLastUsed",
                "iam:GetUser",
                "iam:ListAccessKeys",
                "iam:UpdateAccessKey",
                "iam:TagUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        }
    ]
}
```

How IAM users can manage their own access keys

IAM administrators can grant IAM users the permission to self-manage their access keys by attaching the policy described in Permissions required to manage access keys.

With these permissions, IAM user can use the following procedures to create, activate, deactivate, and delete the access keys associated with their username.

Topics

- Create an access key for yourself (console)

- Deactivate your access key (console)
- Activate your access key (console)
- Delete your access key (console)

Create an access key for yourself (console)

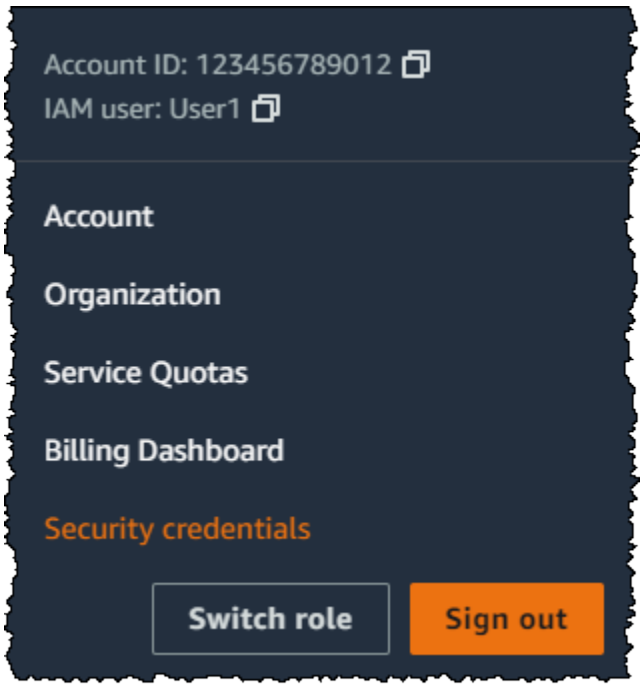
If you have been granted the appropriate permissions you can use the AWS Management Console to create access keys for yourself.

To create your own access keys (console)

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

   > **Note**
   >
   > For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

   To get your AWS account ID, contact your administrator.
2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. In the **Access keys** section, choose **Create access key**. If you already have two access keys, this button is deactivated and you must delete an access key before you can create a new one.
4. On the **Access key best practices & alternatives** page, choose your use case to learn about additional options which can help you avoid creating a long-term access key. If you determine that your use case still requires an access key, choose **Other** and then choose **Next**.
5. (Optional) Set a description tag value for the access key. This adds a tag key-value pair to your IAM user. This can help you identify and update access keys later. The tag key is set to the access key id. The tag value is set to the access key description that you specify. When you are finished, choose **Create access key**.
6. On the **Retrieve access keys** page, choose either **Show** to reveal the value of your user's secret access key, or **Download .csv file**. This is your only opportunity to save your secret access key. After you've saved your secret access key in a secure location, choose **Done**.

Deactivate your access key (console)

If you have been granted the appropriate permissions you can use the AWS Management Console to deactivate your access key.

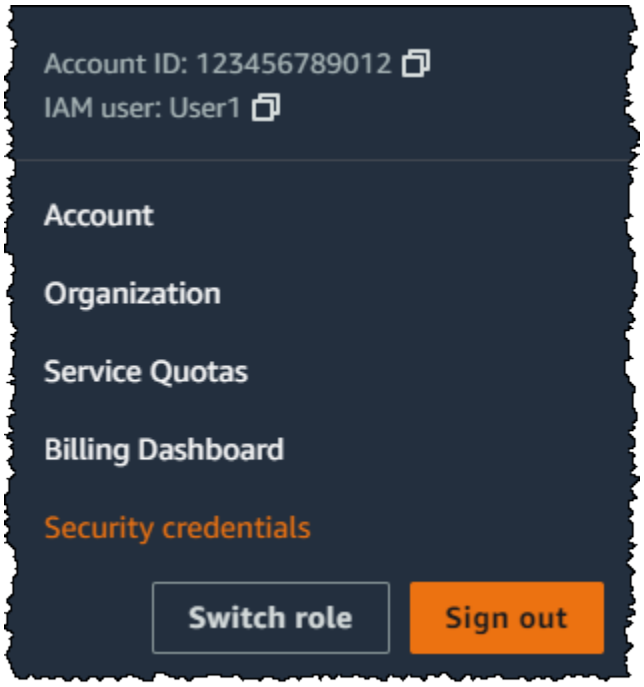
To deactivate an access key

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

> **Note**
>
> For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. In the **Access keys** section find the key you want to deactivate, then choose **Actions**, then choose **Deactivate**. When prompted for confirmation, choose **Deactivate**. A deactivated access key still counts toward your limit of two access keys.

Activate your access key (console)

If you have been granted the appropriate permissions you can use the AWS Management Console to activate your access key.

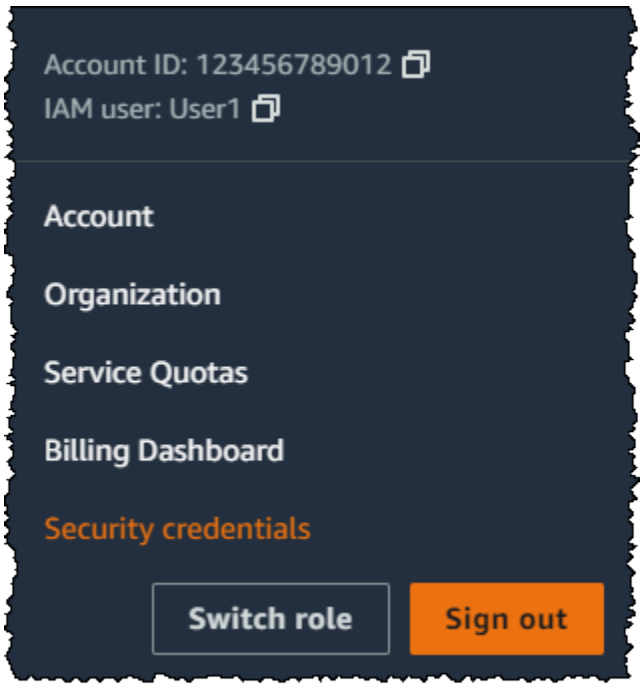
To activate an access key

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

   > **Note**
   >
   > For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

   To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. In the **Access keys** section, find the key to activate, then choose **Actions**, then choose **Activate**.

Delete your access key (console)

If you have been granted the appropriate permissions you can use the AWS Management Console to delete your access key.

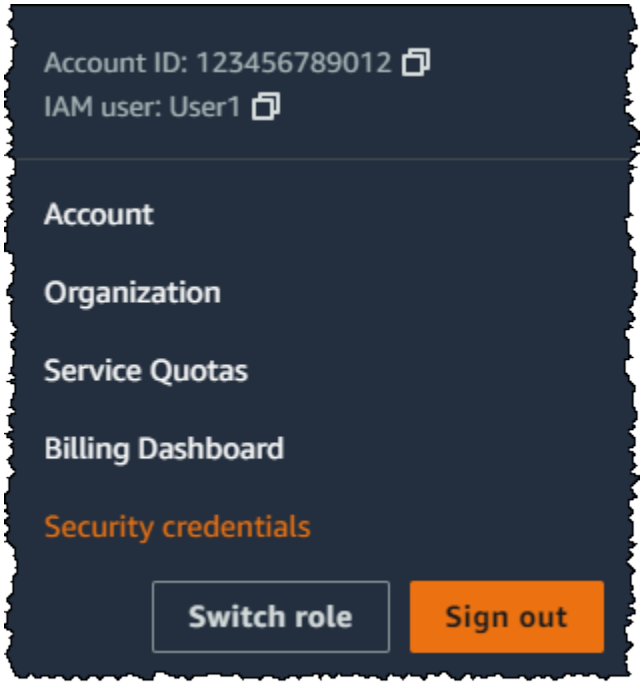
To delete an access key when you no longer need it

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

   > **Note**
   >
   > For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

   To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. In the **Access keys** section, find the key you want to delete, then choose **Actions**, then choose **Delete**. Follow the instructions in the dialog to first **Deactivate** and then confirm the deletion. We recommend that you verify that the access key is no longer in use before you permanently delete it.

How an IAM administrator can manage IAM user access keys

IAM administrators can create, activate, deactivate, and delete the access keys associated with individual IAM users. They can also list the IAM users in the account which have access keys and locate which IAM user has a specific access key.

Topics

- To create an access key for an IAM user
- To deactivate an access key for an IAM user
- To activate an access key for an IAM user
- To delete an access key for an IAM user
- To list the access keys for an IAM user
- To display all the access key IDs for users in your account
- To use an access key ID to find a user
- To find the most recent use of an access key ID

To create an access key for an IAM user

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the user name to go to the user details page.
4. On **Security credentials** tab, in the **Access keys** section, choose **Create access key**.

   If the button is deactivated, then you must delete one of the existing keys before you can create a new one.

5. On the **Access key best practices & alternatives** page, review the best practices and alternatives. Choose your use case to learn about additional options which can help you avoid creating a long-term access key.
6. If you determine that your use case still requires an access key, choose **Other** and then choose **Next**.
7. **(Optional)** On the **Set description tag** page, you can add a description tag to the access key to help track your access key. Select **Create access key**.
8. On the **Retrieve access key page**, choose **Show** to reveal the value of your user's secret access key.
9. To save the access key ID and secret access key to a `.csv` file to a secure location on your computer, choose the **Download .csv file** button.

> **Important**
>
> This is your only time to view or download the newly created access key and you cannot recover it. Make sure you securely maintain your access key.

When you create an access key for your user, that key pair is active by default, and your user can use the pair right away.

AWS CLI

Run the following command:

- `aws iam create-access-key`

API

Call the following operation:

- `CreateAccessKey`

To deactivate an access key for an IAM user

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the user name to go to the user details page.
4. On **Security credentials** tab, in the **Access keys** section, choose the **Actions** drop-down menu, then choose **Deactivate**.
5. In the **Deactivate** dialog box, confirm that you want to deactivate the access key by selecting **Deactivate**

After an access key is deactivated, it can no longer be used by API calls. You can activate it again if needed.

AWS CLI

Run the following command:

- `aws iam update-access-key`

API

Call the following operation:

- `UpdateAccessKey`

To activate an access key for an IAM user

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the user name to go to the user details page.
4. On **Security credentials** tab, in the **Access keys** section, choose the **Actions** drop-down menu, then choose **Activate**.

After an access key is activated, it can be used by API calls. You can deactivate it again if needed.

AWS CLI

Run the following command:

- `aws iam update-access-key`

API

Call the following operation:

- `UpdateAccessKey`

To delete an access key for an IAM user

After an access key has been deactivated, if it is no longer required, delete it.

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the user name to go to the user details page.
4. On **Security credentials** tab, in the **Access keys** section, choose the **Actions** drop-down menu for the inactive access key, then choose **Delete**.
5. In the **Delete** dialog box, confirm that you want to delete the access key by entering the access key ID in the text input field and then selecting **Delete**.

After an access key is deleted, it can't be recovered.

AWS CLI

Run the following command:

- `aws iam delete-access-key`

API

Call the following operation:

- `DeleteAccessKey`

To list the access keys for an IAM user

You can view a list of the access key IDs associated with an IAM user.

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the user name to go to the user details page.
4. On **Security credentials** tab, the **Access keys** section lists the access key IDs for the user including the status of each key displayed.

> **Note**
>
> Only the user's access key ID is visible. The secret access key can only be retrieved when the key is created.

Each IAM user can have two access keys.

AWS CLI

Run the following command:

- `aws iam list-access-keys`

API

Call the following operation:

- `ListAccessKeys`

To display all the access key IDs for users in your account

You can view a list of the access key IDs for users in your AWS account.

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the user name to go to the user details page.
4. If necessary, add the **Access key ID** column to the users table by completing the following steps:

   a. Above the table on the far right, choose the **Preferences** icon ( ).

   b. In the **Preferences** dialog box, under **Select visible columns** turn on **Access key ID**.

   c. Choose **Confirm** to return to the list of users. The list is updated to include the access key ID.
5. The **Access key ID** column shows the state of each access key, followed by its ID; for example, **`Active - AKIAIOSFODNN7EXAMPLE`** or **`Inactive - AKIAI44QH8DHBEXAMPLE`**.

   You can use this information to view and copy the access keys IDs for users with one or two access keys. The column displays **-** for users with no access key.

   > **Note**
   >
   > The secret access key can only be retrieved when the key is created.

Each IAM user can have two access keys.

To use an access key ID to find a user

You can use an access key ID to find a user in your AWS account.

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, in the search box, enter the **Access key ID**, for example AKIAI44QH8DHBEXAMPLE.
3. The IAM user that the access key ID is associated with appears in the navigation pane. Choose the user name to go to the user details page.

To find the most recent use of an access key ID

The most recent use of an access key is displayed in the user's list on the IAM users page, on the user detail page, and is part of the credential report.

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In users list, see the **Access key last used** column.

   If the column is not displayed, choose the **Preferences** icon
   ()
   and under **Select visible columns** turn on **Access key last used** to display the column.
3. (optional) In the navigation pane, under **Access reports**, select **Credential report** to download a report that includes the access key last used information for all of the IAM users in your account.
4. (optional) Select the IAM user to view the user details. The **Summary** section includes the access key IDs, their status, and when they were last used.

AWS CLI

Run the following command:

- `aws iam get-access-key-last-used`

API

Call the following operation:

- GetAccessKeyLastUsed

Update access keys

As a security best practice, we recommend that you update IAM user access keys when needed, such as when an employee leaves your company. IAM users can update their own access keys if they have been granted the necessary permissions.

For details about granting IAM users permissions to update their own access keys, see AWS: Allows IAM users to manage their own password, access keys, and SSH public keys on the Security credentials page. You can also apply a password policy to your account to require that all of your IAM users periodically update their passwords and how often they must do so. For more information, see Set an account password policy for IAM users.

> **Note**
>
> If you lose your secret access key, you must delete the access key and create a new one. The secret access key can be retrieved only at the time you create it. Use this procedure to deactivate and then replace any lost access keys with new credentials.

Topics

- Updating IAM user access keys (console)
- Updating access keys (AWS CLI)
- Updating access keys (AWS API)

Updating IAM user access keys (console)

You can update access keys from the AWS Management Console.

To update access keys for an IAM user without interrupting your applications (console)

1. While the first access key is still active, create a second access key.

a. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

b. In the navigation pane, choose **Users**.

c. Choose the name of the intended user, and then choose the **Security credentials** tab.

d. In the **Access keys** section, choose **Create access key**. On the **Access key best practices & alternatives** page, choose **Other**, then choose **Next**.

e. (Optional) Set a description tag value for the access key to add a tag key-value pair to this IAM user. This can help you identify and update access keys later. The tag key is set to the access key id. The tag value is set to the access key description that you specify. When you are finished, choose **Create access key**.

f. On the **Retrieve access keys** page, choose either **Show** to reveal the value of your user's secret access key, or **Download .csv file**. This is your only opportunity to save your secret access key. After you've saved your secret access key in a secure location, choose **Done**.

   When you create an access key for your user, that key pair is active by default, and your user can use the pair right away. At this point, the user has two active access keys.

2. Update all applications and tools to use the new access key.
3. Determine whether the first access key is still in use by reviewing the **Last used** information for the oldest access key. One approach is to wait several days and then check the old access key for any use before proceeding.
4. Even if the **Last used** information indicates that the old key has never been used, we recommend that you do not immediately delete the first access key. Instead, choose **Actions** and then choose **Deactivate** to deactivate the first access key.
5. Use only the new access key to confirm that your applications are working. Any applications and tools that still use the original access key will stop working at this point because they no longer have access to AWS resources. If you find such an application or tool, you can reactivate the first access key. Then return to Step 3 and update this application to use the new key.
6. After you wait some period of time to make sure that all applications and tools have been updated, you can delete the first access key:

   a. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

   b. In the navigation pane, choose **Users**.

   c. Choose the name of the intended user, and then choose the **Security credentials** tab.

d. In the **Access keys** section for the access key you want to delete, choose **Actions**, and then choose **Delete**. Follow the instructions in the dialog to first **Deactivate** and then confirm the deletion.

To determine which access keys need to be updated or deleted (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. If necessary, add the **Access key age** column to the users table by completing the following steps:

   a. Above the table on the far right, choose the settings icon ().

   b. In **Manage columns**, select **Access key age**.

   c. Choose **Close** to return to the list of users.
4. The **Access key age** column shows the number of days since the oldest active access key was created. You can use this information to find users with access keys that might need to be updated or deleted. The column displays **None** for users with no access key.

Updating access keys (AWS CLI)

You can update access keys from the AWS Command Line Interface.

To update access keys without interrupting your applications (AWS CLI)

1. While the first access key is still active, create a second access key, which is active by default. Run the following command:

   - `aws iam create-access-key`

   At this point, the user has two active access keys.
2. Update all applications and tools to use the new access key.
3. Determine whether the first access key is still in use by using this command:

   - `aws iam get-access-key-last-used`

One approach is to wait several days and then check the old access key for any use before proceeding.

4. Even if step Step 3 indicates no use of the old key, we recommend that you do not immediately delete the first access key. Instead, change the state of the first access key to `Inactive` using this command:

   - `aws iam update-access-key`

5. Use only the new access key to confirm that your applications are working. Any applications and tools that still use the original access key will stop working at this point because they no longer have access to AWS resources. If you find such an application or tool, you can switch its state back to `Active` to reactivate the first access key. Then return to step Step 2 and update this application to use the new key.
6. After you wait some period of time to make sure that all applications and tools have been updated, you can delete the first access key with this command:

   - `aws iam delete-access-key`

Updating access keys (AWS API)

You can update access keys using the AWS API.

To update access keys without interrupting your applications (AWS API)

1. While the first access key is still active, create a second access key, which is active by default. Call the following operation:

   - `CreateAccessKey`

   At this point, the user has two active access keys.
2. Update all applications and tools to use the new access key.
3. Determine whether the first access key is still in use by calling this operation:

   - `GetAccessKeyLastUsed`

   One approach is to wait several days and then check the old access key for any use before proceeding.

4. Even if step Step 3 indicates no use of the old key, we recommend that you do not immediately delete the first access key. Instead, change the state of the first access key to `Inactive` by calling this operation:

   - `UpdateAccessKey`

5. Use only the new access key to confirm that your applications are working. Any applications and tools that still use the original access key will stop working at this point because they no longer have access to AWS resources. If you find such an application or tool, you can switch its state back to `Active` to reactivate the first access key. Then return to step Step 2 and update this application to use the new key.
6. After you wait some period of time to make sure that all applications and tools have been updated, you can delete the first access key by calling this operation:

   - `DeleteAccessKey`

Secure access keys

Anyone who has your access keys has the same level of access to your AWS resources that you do. Consequently, AWS goes to significant lengths to protect your access keys, and, in keeping with our shared-responsibility model, you should as well.

Expand the following sections for guidance to help you protect your access keys.

> **Note**
>
> Your organization might have different security requirements and policies than those described in this topic. The suggestions provided here are intended as general guidelines.

Remove (or don't generate) AWS account root user access keys

One of the best ways to protect your account is to not have access keys for your AWS account root user. Unless you must have root user access keys (which is rare), it is best not to generate them. Instead, create an administrative user in AWS IAM Identity Center for daily administrative tasks.For information about how to create an administrative user in IAM Identity Center, see Getting started in the *IAM Identity Center User Guide*.

If you already have root user access keys for your account, we recommend the following: Find places in your applications where you are currently using access keys (if any), and replace the root

user access keys with IAM user access keys. Then disable and remove the root user access keys. For more information about how to update access keys, see Update access keys

Use temporary security credentials (IAM roles) instead of long-term access keys

In many scenarios, you don't need long-term access keys that never expire (as you have with an IAM user). Instead, you can create IAM roles and generate temporary security credentials. Temporary security credentials consist of an access key ID and a secret access key, but they also include a security token that indicates when the credentials expire.

Long-term access keys, such as those associated with IAM users and the root user, remain valid until you manually revoke them. However, temporary security credentials obtained through IAM roles and other features of the AWS Security Token Service expire after a short period of time. Use temporary security credentials to help reduce your risk in case credentials are accidentally exposed.

Use an IAM role and temporary security credentials in these scenarios:

- **You have an application or AWS CLI scripts running on an Amazon EC2 instance.** Don't use access keys directly in your application. Don't pass access keys to the application, embed them in the application, or let the application read access keys from any source. Instead, define an IAM role that has appropriate permissions for your application and launch the Amazon Elastic Compute Cloud (Amazon EC2) instance with roles for EC2. Doing this associates an IAM role with the Amazon EC2 instance. This practice also enables the application to get temporary security credentials that it can in turn use to make programmatic calls to AWS. The AWS SDKs and the AWS Command Line Interface (AWS CLI) can get temporary credentials from the role automatically.
- **You need to grant cross-account access.** Use an IAM role to establish trust between accounts, and then grant users in one account limited permissions to access the trusted account. For more information, see IAM tutorial: Delegate access across AWS accounts using IAM roles.
- **You have a mobile app.** Don't embed access keys with the app, even in encrypted storage. Instead, use Amazon Cognito to manage user identities in your app. This service lets you authenticate users using Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC)–compatible identity provider. You can then use the Amazon Cognito credentials provider to manage credentials that your app uses to make requests to AWS.
- **You want to federate into AWS and your organization supports SAML 2.0.** If you work for an organization that has an identity provider that supports SAML 2.0, configure the provider to use SAML. You can use SAML to exchange authentication information with AWS and get back a set of temporary security credentials. For more information, see SAML 2.0 federation.

- **You want to federate into AWS and your organization has an on-premises identity store.** If users can authenticate inside your organization, you can write an application that can issue them temporary security credentials for access to AWS resources. For more information, see Enable custom identity broker access to the AWS console.
- **Use conditions in IAM policies to only allow access from expected networks.** You can limit where and how your access keys are used by implementing IAM policies with conditions that specify and allow only expected networks, such as your public IP addresses or Virtual Private Clouds (VPCs). This way you know access keys can only be used from expected and acceptable networks.

> **Note**
>
> Are you using an Amazon EC2 instance with an application that requires programmatic access to AWS resources? If so, use IAM roles for EC2.

Manage IAM user access keys properly

If you must create access keys for programmatic access to AWS, create them for IAM users, granting the users only the permissions they require.

Observe these precautions to help protect IAM user access keys:

- **Don't embed access keys directly into code.** The AWS SDKs and the AWS Command Line Tools enable you to put access keys in known locations so that you don't have to keep them in code.

  Put access keys in one of the following locations:

  - **The AWS credentials file.** The AWS SDKs and AWS CLI automatically use the credentials that you store in the AWS credentials file.

    For information about using the AWS credentials file, see the documentation for your SDK. Examples include Set AWS Credentials and Region in the *AWS SDK for Java Developer Guide* and Configuration and credential files in the *AWS Command Line Interface User Guide*.

    To store credentials for the AWS SDK for .NET and the AWS Tools for Windows PowerShell, we recommend that you use the SDK Store. For more information, see Using the SDK Store in the *AWS SDK for .NET Developer Guide*.

- **Environment variables.** On a multi-tenant system, choose user environment variables, not system environment variables.

  For more information about using environment variables to store credentials, see Environment Variables in the *AWS Command Line Interface User Guide*.

- **Use different access keys for different applications.** Do this so that you can isolate the permissions and revoke the access keys for individual applications if they are exposed. Having separate access keys for different applications also generates distinct entries in AWS CloudTrail log files. This configuration makes it easier for you to determine which application performed specific actions.
- **Update access keys when needed.** If there is a risk that the access key could be compromised, update the access key and delete the previous access key. For details, see Update access keys
- **Remove unused access keys.** If a user leaves your organization, remove the corresponding IAM user so that the user can no longer access your resources. To find out when an access key was last used, use the `GetAccessKeyLastUsed` API (AWS CLI command: `aws iam get-access-key-last-used`).
- **Use temporary credentials and configure multi-factor authentication for your most sensitive API operations.** With IAM policies, you can specify which API operations a user is allowed to call. In some cases, you might want the additional security of requiring users to be authenticated with AWS MFA before you allow them to perform particularly sensitive actions. For example, you might have a policy that allows a user to perform the Amazon EC2 `RunInstances`, `DescribeInstances`, and `StopInstances` actions. But you might want to restrict a destructive action like `TerminateInstances` and make sure that users can perform that action only if they authenticate with an AWS MFA device. For more information, see Secure API access with MFA.

Access the mobile app using AWS access keys

You can access a limited set of AWS services and features using the AWS mobile app. The mobile app helps you support incident response while on the go. For more information and to download the app, see AWS Console Mobile Application.

You can sign in to the mobile app using your console password or your access keys. As a best practice, do not use root user access keys. Instead, we strongly recommend that in addition to using a password or biometric lock on your mobile device, you create an IAM user specifically for

managing AWS resources using the mobile app. If you lose your mobile device, you can remove the IAM user's access.

To sign in using access keys (mobile app)

1. Open the app on your mobile device.
2. If this is the first time that you're adding an identity to the device, choose **Add an identity** and then choose **Access keys**.

   If you have already signed in using another identity, choose the menu icon and choose **Switch identity**. Then choose **Sign in as a different identity** and then **Access keys**.
3. On the **Access keys** page, enter your information:

   - **Access key ID** – Enter your access key ID.
   - **Secret access key** – Enter your secret access key.
   - **Identity name** – Enter the name of the identity that will appear in the mobile app. This does not need to match your IAM user name.
   - **Identity PIN** – Create a personal identification number (PIN) that you will use for future sign-ins.

   > **Note**
   >
   > If you enable biometrics for the AWS mobile app, you will be prompted to use your fingerprint or facial recognition for verification instead of the PIN. If the biometrics fail, you might be prompted for the PIN instead.
4. Choose **Verify and add keys**.

   You can now access a select set of your resources using the mobile app.

Related information

The following topics provide guidance for setting up the AWS SDKs and the AWS CLI to use access keys:

- Set AWS credentials and Region in the *AWS SDK for Java Developer Guide*
- Using the SDK Store in the *AWS SDK for .NET Developer Guide*
- Providing Credentials to the SDK in the *AWS SDK for PHP Developer Guide*

- Configuration in the Boto 3 (AWS SDK for Python) documentation
- Using AWS Credentials in the *AWS Tools for Windows PowerShell User Guide*
- Configuration and credential files in the *AWS Command Line Interface User Guide*
- Granting access using an IAM role in the *AWS SDK for .NET Developer Guide*
- Configure IAM roles for Amazon EC2 in the *AWS SDK for Java 2.x*

Using access keys and secret key credentials for console access

It is possible to use access key and secret key credentials for direct AWS Management Console access, not just the AWS CLI. This can be achieved using the AWS STS `GetFederationToken` API call. By constructing a console URL using the temporary credentials and token provided by `GetFederationToken`, IAM principals can access the console. For more information, see Enable custom identity broker access to the AWS console.

It is worth clarifying that when signing into the console directly using IAM or root user credentials with MFA enabled, MFA will be required. However, if the method described above (using temporary credentials with `GetFederationToken`) is used, MFA will NOT be required.

Auditing access keys

You can review the AWS access keys in your code to determine whether the keys are from an account that you own. You can pass an access key ID using the `aws sts get-access-key-info` AWS CLI command or the `GetAccessKeyInfo` AWS API operation.

The AWS CLI and AWS API operations return the ID of the AWS account to which the access key belongs. Access key IDs beginning with `AKIA` are long-term credentials for an IAM user or an AWS account root user. Access key IDs beginning with `ASIA` are temporary credentials that are created using AWS STS operations. If the account in the response belongs to you, you can sign in as the root user and review your root user access keys. Then, you can pull a credentials report to learn which IAM user owns the keys. To learn who requested the temporary credentials for an `ASIA` access key, view the AWS STS events in your CloudTrail logs.

For security purposes, you can review AWS CloudTrail logs to learn who performed an action in AWS. You can use the `sts:SourceIdentity` condition key in the role trust policy to require users to specify an identity when they assume a role. For example, you can require that IAM users specify their own user name as their source identity. This can help you determine which user performed a specific action in AWS. For more information, see `sts:SourceIdentity`.

This operation does not indicate the state of the access key. The key might be active, inactive, or deleted. Active keys might not have permissions to perform an operation. Providing a deleted access key might return an error that the key doesn't exist.

AWS Multi-factor authentication in IAM

For increased security, we recommend that you configure multi-factor authentication (MFA) to help protect your AWS resources. You can enable MFA for the AWS account root user of all AWS accounts, including standalone accounts, management accounts, and member accounts, as well as for your IAM users. We recommend that you use phishing-resistant MFA such as passkeys and security keys whenever possible. These FIDO-based authenticators use public key cryptography and are resistant to phishing, man-in-the-middle, and replay attacks, providing a stronger level of security than TOTP-based options.

MFA is enforced for all account types for their root user. For more information, see Secure your AWS Organizations account root user credentials.

When you enable MFA for the root user, it affects only the root user credentials. IAM users in the account are distinct identities with their own credentials, and each identity has its own MFA configuration. For more information about using MFA to protect the root user, see Multi-factor authentication for AWS account root user.

Your AWS account root user and IAM users can register up to eight MFA devices of any type. Registering multiple MFA devices can provide flexibility and help you reduce the risk of access interruption if a device is lost or broken. You only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI.

> **Note**
>
> We recommend that you require your human users to use temporary credentials when accessing AWS. Have you considered using AWS IAM Identity Center? You can use IAM Identity Center to centrally manage access to multiple AWS accounts and provide users with MFA-protected, single sign-on access to all their assigned accounts from one place. With IAM Identity Center, you can create and manage user identities in IAM Identity Center or easily connect to your existing SAML 2.0 compatible identity provider. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

MFA adds extra security that requires users to provide unique authentication from an AWS supported MFA mechanism in addition to their sign-in credentials when they access AWS websites or services.

MFA types

AWS supports the following MFA types:

Contents

- Passkeys and security keys
- Virtual authenticator applications
- Hardware TOTP tokens

Passkeys and security keys

AWS Identity and Access Management supports passkeys and security keys for MFA. Based on FIDO standards, passkeys use public key cryptography to provide strong, phishing-resistant authentication that is more secure than passwords. AWS supports two types of passkeys: device-bound passkeys (security keys) and synced passkeys.

- **Security keys**: These are physical devices, like a YubiKey, used as a second factor for authentication. A single security key can support multiple root user accounts and IAM users.
- **Synced passkeys**: These use credential managers from providers such as Google, Apple, Microsoft accounts, and third-party services like 1Password, Dashlane, and Bitwarden as a second factor.

You can use built-in biometric authenticators, like Touch ID on Apple MacBooks, to unlock your credential manager and sign in to AWS. Passkeys are created with your chosen provider using your fingerprint, face, or device PIN. You can also use a cross-device authentication (CDA) passkey from one device, like a mobile device or hardware security key, to sign in on another device like a laptop. For more information, see cross-device authentication (CDA).

You can sync passkeys across your devices to facilitate sign-ins with AWS, enhancing usability and recoverability. For more information about enabling passkeys and security keys, see Enable a passkey or security key for the root user (console).

The FIDO Alliance maintains a list of all FIDO Certified products that are compatible with FIDO specifications.

Virtual authenticator applications

A virtual authenticator application runs on a phone or other device and emulates a physical device. Virtual authenticator apps implement the time-based one-time password (TOTP) algorithm and support multiple tokens on a single device. The user must type a valid code from the device when prompted during sign-in. Each token assigned to a user must be unique. A user can't type a code from another user's token to authenticate.

We recommend that you use phishing-resistant MFA such as passkeys or security keys for the strongest protection. If you are not yet able to use passkeys or security keys, we recommend that you use a virtual MFA device as an interim measure while waiting for hardware purchase approval or while you wait for your hardware to arrive. For a list of a few supported apps that you can use as virtual MFA devices, see Multi-Factor Authentication (MFA).

For instructions on setting up a virtual MFA device for an IAM user, see Assign a virtual MFA device in the AWS Management Console.

> **Note**
>
> Unassigned virtual MFA devices in your AWS account are deleted when you're adding new virtual MFA devices either via the AWS Management Console or during the sign-in process. Unassigned virtual MFA devices are devices in your account but not used by account root user or IAM users for the sign-in process. They're deleted so new virtual MFA devices can be added to your account. It also allows you to reuse device names.
>
> - To view unassigned virtual MFA devices in your account, you can use either the list-virtual-mfa-devices AWS CLI command or API call.
> - To deactivate a virtual MFA device, you can use either the deactivate-mfa-device AWS CLI command or API call. The device will become unassigned.
> - To attach an unassigned virtual MFA device to your AWS account root user or IAM users, you'll need the authentication code generated by the device along with either the enable-mfa-device AWS CLI command or API call.

Hardware TOTP tokens

A hardware device generates a six-digit numeric code based on the time-based one-time password (TOTP) algorithm. The user must type a valid code from the device on a second webpage during sign-in.

These tokens are used exclusively with AWS accounts. You can only use tokens that have their unique token seeds shared securely with AWS. Token seeds are secret keys generated at the time of token production. Tokens purchased from other sources will not function with IAM. To ensure compatibility, you must purchase your hardware MFA device from one of the following links: OTP token or OTP display card.

- Each MFA device assigned to a user must be unique. A user cannot type a code from another user's device to be authenticated. For information on supported hardware MFA devices, see Multi-Factor Authentication (MFA).
- If you want to use a physical MFA device, we recommend that you use security keys as an alternative to hardware TOTP devices. Security keys have no battery requirements, are phishing resistant, and support multiple users on a single device.

You can enable a passkey or security key from the AWS Management Console only, not from the AWS CLI or AWS API. Before you can enable a security key, you must have physical access to the device.

For instructions on setting up a hardware TOTP token for an IAM user, see Assign a hardware TOTP token in the AWS Management Console.

> **Note**
>
> **SMS text message-based MFA** – AWS ended support for enabling SMS multi-factor authentication (MFA). We recommend that customers who have IAM users that use SMS text message-based MFA switch to one of the following alternative methods: Passkey or security key, virtual (software-based) MFA device, or hardware MFA device. You can identify the users in your account with an assigned SMS MFA device. In the IAM console, choose **Users** from the navigation pane, and look for users with **SMS** in the **MFA** column of the table.

MFA recommendations

To help secure your AWS identities, follow these recommendations for MFA authentication.

- We recommend that you use phishing-resistant MFA, such as passkeys and security keys, as your MFA device. These FIDO-based authenticators provide the strongest protection against attacks such as phishing.

- We recommend that you enable multiple MFA devices to the AWS account root user and IAM users in your AWS accounts. This allows you to raise the security bar in your AWS accounts and simplify managing access to highly privileged users, such as the AWS account root user.
- You can register up to **eight** MFA devices of any combination of the currently supported MFA types with your AWS account root user and IAM users. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI as that user. An IAM user must authenticate with an existing MFA device to enable or disable an additional MFA device.
- In the event of a lost, stolen, or inaccessible MFA device you can use one of the remaining MFA devices to access the AWS account without performing the AWS account recovery procedure. If an MFA device is lost or stolen, it should be disassociated from the IAM principal with which it is associated.
- The use of multiple MFAs allows your employees in geographically dispersed locations or working remotely to use hardware-based MFA to access AWS without having to coordinate the physical exchange of a single hardware device between employees.
- The use of additional MFA devices for IAM principals allows you to use one or more MFAs for everyday usage, while also maintaining physical MFA devices in a secure physical location such as a vault or safe for backup and redundancy.

Notes

- You cannot pass the MFA information for a security key or passkey to AWS STS API operations to request temporary credentials. You can obtain credentials for use with the AWS CLI and AWS SDKs when using a security key or passkey by running the `aws login` command.
- You cannot use AWS CLI commands or AWS API operations to enable FIDO security keys.
- You cannot use the same name for more than one root user or IAM MFA device.

Additional resources

The following resources can help you learn more about MFA.

- For more information about using MFA to access AWS, see MFA enabled sign-in.

- You can use IAM Identity Center to enable secure MFA access to your AWS access portal, IAM Identity Center integrated apps, and the AWS CLI. For more information, see Enable MFA in IAM Identity Center.
- To verify whether MFA is already enabled for your users, see Check MFA status.

Assign a passkey or security key in the AWS Management Console

Passkeys are a type of multi-factor authentication (MFA) device that you can use to protect your AWS resources. AWS supports synced passkeys and device-bound passkeys also known as security keys.

Synced passkeys allow IAM users to access their FIDO sign-in credentials on many of their devices, even new ones, without having to re-enroll every device on every account. Synced passkeys include first-party credential managers like Google, Apple, and Microsoft and third-party credential managers such as 1Password, Dashlane, and Bitwarden as a second factor. You can also use on-device biometrics (e.g., TouchID, FaceID) to unlock your chosen credential manager to use passkeys.

Alternatively, device-bound passkeys are bound to a FIDO security key that you plug into a USB port on your computer and then tap when prompted to securely complete the sign-in process. If you already use a FIDO security key with other services, and it has an AWS supported configuration (for example, the YubiKey 5 Series from Yubico), you can also use it with AWS. Otherwise, you need to purchase a FIDO security key if you want to use WebAuthn for MFA in AWS. Additionally, FIDO security keys can support multiple IAM or root users on the same device, enhancing their utility for account security. For specifications and purchase information for both device types, see Multi-Factor Authentication.

You can register up to **eight** MFA devices of any combination of the currently supported MFA types with your AWS account root user and IAM users. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI as that user. We recommend that you register multiple MFA devices. For example, you can register a built-in authenticator and also register a security key that you keep in a physically secure location. If you're unable to use your built-in authenticator, then you can use your registered security key. For authenticator applications, we also recommend enabling the cloud backup or sync feature in those apps to help you avoid losing access to your account if you lose or break your device with the authenticator apps.

> **Note**
>
> We recommend that you require your human users to use temporary credentials when accessing AWS. Your users can federate into AWS with an identity provider where they authenticate with their corporate credentials and MFA configurations. To manage access to AWS and business applications, we recommend that you use IAM Identity Center. For more information, see the IAM Identity Center User Guide.

Topics

- Permissions required
- Enable a passkey or security key for your own IAM user (console)
- Enable a passkey or security key for another IAM user (console)
- Replace a passkey or security key
- Supported configurations for using passkeys and security keys

Permissions required

To manage a FIDO passkey for your own IAM user while protecting sensitive MFA-related actions, you must have the permissions from the following policy:

> **Note**
>
> The ARN values are static values and are not an indicator of what protocol was used to register the authenticator. We have deprecated U2F, so all new implementations use WebAuthn.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowManageOwnUserMFA",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:DeactivateMFADevice",
                "iam:EnableMFADevice",
                "iam:GetUser",
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "DenyAllExceptListedIfNoMFA",
            "Effect": "Deny",
            "NotAction": [
                "iam:EnableMFADevice",
                "iam:GetUser",
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "*",
            "Condition": {
                "BoolIfExists": {
                    "aws:MultiFactorAuthPresent": "false"
                }
            }
        }
    ]
}
```

Enable a passkey or security key for your own IAM user (console)

You can enable a passkey or security key for your own IAM user from the AWS Management Console only, not from the AWS CLI or AWS API. Before you can enable a security key, you must have physical access to the device.

To enable a passkey or security key for your own IAM user (console)

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

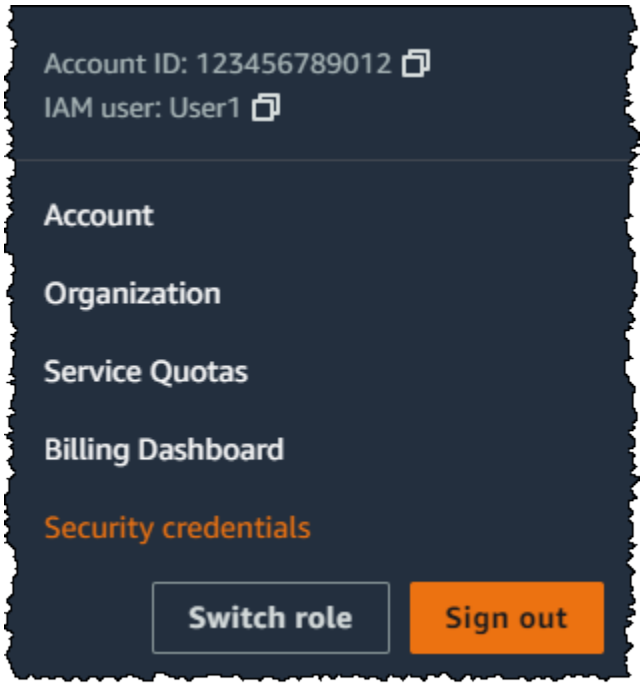
   Note

   For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user,

choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. On the selected IAM user's page, choose the **Security credentials** tab.
4. Under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
5. On the **MFA device name** page, enter a **Device name**, choose **Passkey or Security Key**, and then choose **Next**.
6. On **Set up device**, set up your passkey. Create a passkey with biometric data like your face or fingerprint, with a device pin, or by inserting the FIDO security key into your computer's USB port and tapping it.
7. Follow the instructions on your browser and then choose **Continue**.

You have now registered your passkey or security key for use with AWS. For information about using MFA with the AWS Management Console, see MFA enabled sign-in.

Enable a passkey or security key for another IAM user (console)

You can enable a passkey or security key for another IAM user from the AWS Management Console only, not from the AWS CLI or AWS API.

To enable a passkey or security key for another IAM user (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Under **Users**, choose the name of the user for whom you want to enable MFA.
4. On the selected IAM user page, choose the **Security Credentials** tab.
5. Under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
6. On the **MFA device name** page, enter a **Device name**, choose **Passkey or Security Key**, and then choose **Next**.
7. On **Set up device**, set up your passkey. Create a passkey with biometric data like your face or fingerprint, with a device pin, or by inserting the FIDO security key into your computer's USB port and tapping it.
8. Follow the instructions on your browser and then choose **Continue**.

You have now registered a passkey or security key for another IAM user to use with AWS. For information about using MFA with the AWS Management Console, see MFA enabled sign-in.

Replace a passkey or security key

You can have up to eight MFA devices of any combination of the currently supported MFA types assigned to a user at a time with your AWS account root user and IAM users. If the user loses a FIDO authenticator or needs to replace it for any reason, you must first deactivate the old FIDO authenticator. Then you can add a new MFA device for the user.

- To deactivate the device currently associated with an IAM user, see Deactivate an MFA device.
- To add a new FIDO security key for an IAM user, see Enable a passkey or security key for your own IAM user (console).

If you don't have access to a new passkey or security key, you can enable a new virtual MFA device or hardware TOTP token. See one of the following for instructions:

- Assign a virtual MFA device in the AWS Management Console
- Assign a hardware TOTP token in the AWS Management Console

Supported configurations for using passkeys and security keys

You can use FIDO2 device-bound passkeys, also known as security keys, as a multi-factor authentication (MFA) method with IAM using currently supported configurations. These include FIDO2 devices supported by IAM and browsers that support FIDO2.

Before you register your FIDO2 device, check that you're using the latest browser and operating system (OS) version. Features might behave differently across different browsers, authenticators, and OS clients. If your device registration fails on one browser, you can try to register with another browser.

FIDO2 is an open authentication standard and an extension of FIDO U2F, offering the same high level of security based on public key cryptography. FIDO2 consists of the W3C Web Authentication specification (WebAuthn API) and the FIDO Alliance Client-to-Authenticator Protocol (CTAP), an application layer protocol. CTAP enables communication between client or platform, like a browser or operating system, with an external authenticator.

When you enable a FIDO Certified authenticator in AWS, the security key creates a new key pair for use with only AWS. First, you enter your credentials. When prompted, you tap the security key, which responds to the authentication challenge issued by AWS. To learn more about the FIDO2 standard, see the FIDO2 Project.

FIDO2 devices supported by AWS

IAM supports FIDO2 security devices that connect to your devices through USB, Bluetooth, or NFC. IAM also supports platform authenticators such as TouchID or FaceID. IAM does not support local passkey registration for Windows Hello. To create and use passkeys, Windows users should use cross-device authentication where you use a passkey from one device like a mobile device or hardware security key to sign in on another device like a laptop.

> **Note**
>
> AWS requires access to the physical USB port on your computer to verify your FIDO2 device. Security keys will not work with a virtual machine, a remote connection, or a browser's incognito mode.

The FIDO Alliance maintains a list of all FIDO2 products that are compatible with FIDO specifications.

Browsers that support FIDO2

The availability of FIDO2 security devices that run in a web browser depends on the combination of browser and operating system. The following browsers currently support the use of security keys:

Web browser	macOS 10.15+	Windows 10	Linux	iOS 14.5+	Android 7+
Chrome	Yes	Yes	Yes	Yes	No
Safari	Yes	No	No	Yes	No
Edge	Yes	Yes	No	Yes	No
Firefox	Yes	Yes	No	Yes	No

Note

Most Firefox versions that currently support FIDO2 don't enable support by default. For instructions on enabling FIDO2 support in Firefox, see Troubleshoot Passkeys and FIDO Security Keys.
Firefox on macOS might not fully support cross-device authentication workflows for passkeys. You might get a prompt to touch a security key instead of proceeding with cross-device authentication. We recommend using a different browser, such as Chrome or Safari, for signing in with passkeys on macOS.

For more information about browser support for a FIDO2-Certified device like YubiKey, see Operating system and web browser support for FIDO2 and U2F.

Browser plugins

AWS supports only browsers that natively support FIDO2. AWS doesn't support using plugins to add FIDO2 browser support. Some browser plugins are incompatible with the FIDO2 standard and can cause unexpected results with FIDO2 security keys.

For information on disabling browser plugins and other troubleshooting tips, see I can't enable my FIDO security key.

Device certifications

We capture and assign device-related certifications, such as FIPS validation and FIDO certification level, only during the registration of a security key. Your device certification is retrieved from the FIDO Alliance Metadata Service (MDS).

If the certification status or level of your security key changes, it will not be reflected in the device tags automatically. To update the certification information of a device, register the device again to fetch the updated certification information.

AWS provides the following certification types as condition keys during device registration, obtained from the FIDO MDS: FIPS-140-2, FIPS-140-3, and FIDO certification levels. You can restrict authenticator registration in IAM policies based on certification type and level. For more information, see the policies below.

Example policies for device certifications

The following use cases show sample policies that allow you to register MFA devices with FIPS certifications.

Topics

- Use case 1: Allow registering only devices that have FIPS-140-2 L2 certifications
- Use case 2: Allow registering devices that have FIPS-140-2 L2 and FIDO L1 certifications
- Use case 3: Allow registering devices that have either FIPS-140-2 L2 or FIPS-140-3 L2 certifications
- Use case 4: Allow registering devices that have FIPS-140-2 L2 certification and support other MFA types like virtual authenticators and hardware TOTP

Use case 1: Allow registering only devices that have FIPS-140-2 L2 certifications

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
```

```
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-2-certification": "L2"
                }
            }
        }
    ]
}
```

Use case 2: Allow registering devices that have FIPS-140-2 L2 and FIDO L1 certifications

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
```

```
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-2-certification": "L2",
                    "iam:FIDO-certification": "L1"
                }
            }
        }
    ]
}
```

Use case 3: Allow registering devices that have either FIPS-140-2 L2 or FIPS-140-3 L2 certifications

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-2-certification": "L2"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-3-certification": "L2"
                }
            }
        }
    ]
}
```

Use case 4: Allow registering devices that have FIPS-140-2 L2 certification and support other MFA types like virtual authenticators and hardware TOTP

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:EnableMFADevice",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:RegisterSecurityKey": "Create"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "iam:EnableMFADevice",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:RegisterSecurityKey": "Activate",
          "iam:FIDO-FIPS-140-2-certification": "L2"
        }
      }
```

```
    },
    {
      "Effect": "Allow",
      "Action": "iam:EnableMFADevice",
      "Resource": "*",
      "Condition": {
        "Null": {
          "iam:RegisterSecurityKey": "true"
        }
      }
    }
  ]
}
```

AWS CLI and AWS API

AWS supports using passkeys and security keys only in the AWS Management Console. Using passkeys and security keys for MFA is not supported in the AWS CLI and AWS API, or for access to MFA-protected API operations.

Additional resources

- For more information on using passkeys and security keys in AWS, see Assign a passkey or security key in the AWS Management Console.
- For help with troubleshooting passkeys and security keys in AWS, see Troubleshoot Passkeys and FIDO Security Keys.
- For general industry information on FIDO2 support, see FIDO2 Project.

Assign a virtual MFA device in the AWS Management Console

> **Important**
>
> AWS recommends that you use a passkey or security key for MFA to AWS, wherever possible. For more information, see Assign a passkey or security key in the AWS Management Console.

You can use a phone or other device as a virtual multi-factor authentication (MFA) device. To do this, install a mobile app that is compliant with RFC 6238, a standards-based TOTP (time-based

one-time password) algorithm. These apps generate a six-digit authentication code. Because authenticators can run on unsecured mobile devices, and the codes could potentially be shared with unauthorized parties, TOTP-based MFA does not provide the same level of security as phishing-resistant options such as FIDO2 security keys and passkeys. We recommend that you use passkeys or security keys for MFA for the strongest protection against attacks such as phishing.

If you are not yet able to use passkeys or security keys, we recommend that you use a virtual MFA device as an interim measure while you wait for any hardware purchase approvals or for your hardware to arrive.

Most virtual MFA apps support creating multiple virtual devices, allowing you to use the same app for multiple AWS accounts or users. You can register up to **eight** MFA devices of any combination of MFA types with your AWS account root user and IAM users. You only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI. We recommend that you register multiple MFA devices. For authenticator applications, we also recommend enabling the cloud backup or sync feature to help you avoid losing access to your account if you lose or break your device.

AWS requires a virtual MFA app that produces a six-digit OTP. For a list of virtual MFA apps that you can use, see Multi-Factor Authentication.

Topics

- Permissions required
- Enable a virtual MFA device for an IAM user (console)
- Replace a virtual MFA device

Permissions required

To manage virtual MFA devices for your IAM user, you must have the permissions from the following policy: AWS: Allows MFA-authenticated IAM users to manage their own MFA device on the Security credentials page.

Enable a virtual MFA device for an IAM user (console)

You can use IAM in the AWS Management Console to enable and manage a virtual MFA device for an IAM user in your account. You can attach tags to your IAM resources, including virtual MFA devices, to identify, organize, and control access to them. You can tag virtual MFA devices only when you use the AWS CLI or AWS API. To enable and manage an MFA device using the AWS CLI or

AWS API, see Assign MFA devices in the AWS CLI or AWS API. For more information about tagging IAM resources, see Tags for AWS Identity and Access Management resources.

> **Note**
>
> You must have physical access to the hardware that will host the user's virtual MFA device in order to configure MFA. For example, you might configure MFA for a user who will use a virtual MFA device running on a smartphone. In that case, you must have the smartphone available in order to finish the wizard. Because of this, you might want to let users configure and manage their own virtual MFA devices. In that case, you must grant users the permissions to perform the necessary IAM actions. For more information and for an example of an IAM policy that grants these permissions, see the IAM tutorial: Permit users to manage their credentials and MFA settings and example policy AWS: Allows MFA-authenticated IAM users to manage their own MFA device on the Security credentials page.

To enable a virtual MFA device for an IAM user (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. In the **Users** list, choose the name of the IAM user.
4. Choose the **Security Credentials** tab. Under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Authenticator app**, and then choose **Next**.

   IAM generates and displays configuration information for the virtual MFA device, including a QR code graphic. The graphic is a representation of the "secret configuration key" that is available for manual entry on devices that do not support QR codes.
6. Open your virtual MFA app. For a list of apps that you can use for hosting virtual MFA devices, see Multi-Factor Authentication.

   If the virtual MFA app supports multiple virtual MFA devices or accounts, choose the option to create a new virtual MFA device or account.
7. Determine whether the MFA app supports QR codes, and then do one of the following:

- From the wizard, choose **Show QR code**, and then use the app to scan the QR code. This might be a camera icon or **Scan code** option that uses the device's camera to scan the code.
- From the wizard, choose **Show secret key**, and then type the secret key into your MFA app.

When you are finished, the virtual MFA device starts generating one-time passwords.

8. On the **Set up device** page, in the **MFA code 1** box, type the one-time password that currently appears in the virtual MFA device. Wait up to 30 seconds for the device to generate a new one-time password. Then type the second one-time password into the **MFA code 2** box. Choose **Add MFA**.

> **Important**
>
> Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resync the device.

The virtual MFA device is now ready for use with AWS. For information about using MFA with the AWS Management Console, see MFA enabled sign-in.

> **Note**
>
> Unassigned virtual MFA devices in your AWS account are deleted when you're adding new virtual MFA devices either via the AWS Management Console or during the sign-in process. Unassigned virtual MFA devices are devices in your account but not used by account root user or IAM users for the sign-in process. They're deleted so new virtual MFA devices can be added to your account. It also allows you to reuse device names.
>
> - To view unassigned virtual MFA devices in your account, you can use either the list-virtual-mfa-devices AWS CLI command or API call.
> - To deactivate a virtual MFA device, you can use either the deactivate-mfa-device AWS CLI command or API call. The device will become unassigned.

- To attach an unassigned virtual MFA device to your AWS account root user or IAM users, you'll need the authentication code generated by the device along with either the enable-mfa-device AWS CLI command or API call.

Replace a virtual MFA device

Your AWS account root user and IAM users can register up to **eight** MFA devices of any combination of MFA types. If the user loses a device or needs to replace it for any reason, deactivate the old device. Then you can add the new device for the user.

- To deactivate the device currently associated with another IAM user, see Deactivate an MFA device.
- To add a replacement virtual MFA device for another IAM user, follow the steps in the procedure Enable a virtual MFA device for an IAM user (console) above.
- To add a replacement virtual MFA device for the AWS account root user, follow the steps in the procedure Enable a virtual MFA device for the root user (console).

Assign a hardware TOTP token in the AWS Management Console

Important

AWS recommends that you use a passkey or security key for MFA to AWS, wherever possible. For more information, see Assign a passkey or security key in the AWS Management Console.

A hardware TOTP token generates a six-digit numeric code based upon a time-based one-time password (TOTP) algorithm. The user must type a valid code from the device when prompted during the sign-in process. Each MFA device assigned to a user must be unique; a user cannot type a code from another user's device to be authenticated. MFA devices cannot be shared across accounts or users.

Hardware TOTP tokens and FIDO security keys are both physical devices that you purchase. Hardware MFA devices generate TOTP codes for authentication when you sign in to AWS. They rely on batteries, which may need replacement and resynchronization with AWS over time. FIDO security keys, which utilize public key cryptography, do not require batteries and offer a seamless

authentication process. We recommend using FIDO security keys for their phishing resistance, which provides a more secure alternative to TOTP devices. Additionally, FIDO security keys can support multiple IAM or root users on the same device, enhancing their utility for account security. For specifications and purchase information for both device types, see Multi-Factor Authentication.

You can enable a hardware TOTP token for an IAM user from the AWS Management Console, the command line, or the IAM API. To enable an MFA device for your AWS account root user, see Enable a hardware TOTP token for the root user (console).

You can register up to **eight** MFA devices of any combination of the currently supported MFA types with your AWS account root user and IAM users. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console or create a session through the AWS CLI as that user.

> **Important**
>
> We recommend that you enable multiple MFA devices for your users for continued access to your account in case of a lost or inaccessible MFA device.

> **Note**
>
> If you want to enable the MFA device from the command line, use `aws iam enable-mfa-device`. To enable the MFA device with the IAM API, use the `EnableMFADevice` operation.

Topics

- Permissions required
- Enable a hardware TOTP token for your own IAM user (console)
- Enable a hardware TOTP token for another IAM user (console)
- Replace a physical MFA device

Permissions required

To manage a hardware TOTP token for your own IAM user while protecting sensitive MFA-related actions, you must have the permissions from the following policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowManageOwnUserMFA",
            "Effect": "Allow",
            "Action": [
                "iam:DeactivateMFADevice",
                "iam:EnableMFADevice",
                "iam:GetUser",
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "DenyAllExceptListedIfNoMFA",
            "Effect": "Deny",
            "NotAction": [
                "iam:EnableMFADevice",
                "iam:GetUser",
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}",
            "Condition": {
                "BoolIfExists": {
                    "aws:MultiFactorAuthPresent": "false"
                }
            }
        }
    ]
}
```

Enable a hardware TOTP token for your own IAM user (console)

You can enable your own hardware TOTP token from the AWS Management Console.

> **Note**
>
> Before you can enable a hardware TOTP token, you must have physical access to the device.

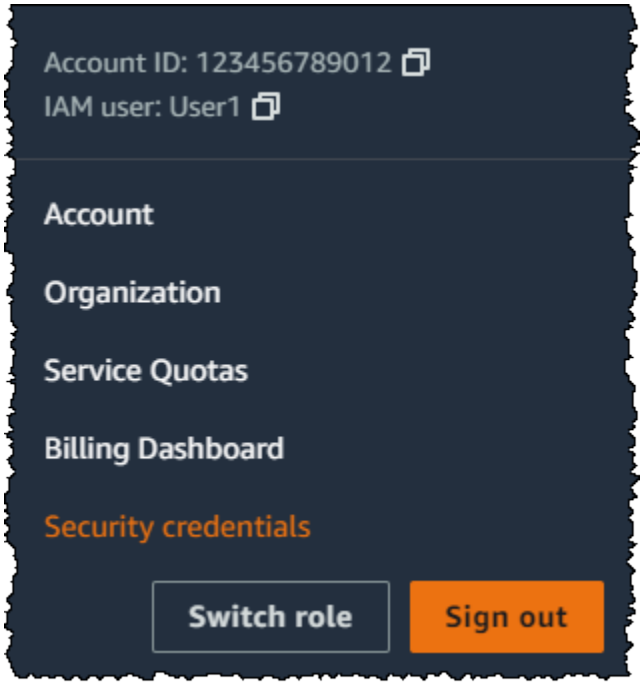
To enable a hardware TOTP token for your own IAM user (console)

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

   > **Note**
   >
   > For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

   To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. On the **AWS IAM credentials** tab, in the **Multi-factor authentication (MFA)** section, choose **Assign MFA device**.
4. In the wizard, type a **Device name**, choose **Hardware TOTP token**, and then choose **Next**.
5. Type the device serial number. The serial number is usually on the back of the device.
6. In the **MFA code 1** box, type the six-digit number displayed by the MFA device. You might need to press the button on the front of the device to display the number.



7. Wait 30 seconds while the device refreshes the code, and then type the next six-digit number into the **MFA code 2** box. You might need to press the button on the front of the device again to display the second number.
8. Choose **Add MFA**.

> **Important**
>
> Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resync the device.

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see MFA enabled sign-in.

Enable a hardware TOTP token for another IAM user (console)

You can enable a hardware TOTP token for another IAM user from the AWS Management Console.

To enable a hardware TOTP token for another IAM user (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the name of the user for whom you want to enable MFA.

4. Choose the **Security Credentials** tab. Under **Multi-factor authentication (MFA)**, choose **Assign MFA device**.
5. In the wizard, type a **Device name**, choose **Hardware TOTP token**, and then choose **Next**.
6. Type the device serial number. The serial number is usually on the back of the device.
7. In the **MFA code 1** box, type the six-digit number displayed by the MFA device. You might need to press the button on the front of the device to display the number.



8. Wait 30 seconds while the device refreshes the code, and then type the next six-digit number into the **MFA code 2** box. You might need to press the button on the front of the device again to display the second number.
9. Choose **Add MFA**.

> **Important**
>
> Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resync the device.

The device is ready for use with AWS. For information about using MFA with the AWS Management Console, see MFA enabled sign-in.

Replace a physical MFA device

You can have up to eight MFA devices of any combination of the currently supported MFA types assigned to a user at a time with your AWS account root user and IAM users. If the user loses a device or needs to replace it for any reason, you must first deactivate the old device. Then you can add the new device for the user.

- To deactivate the device currently associated with a user, see Deactivate an MFA device.
- To add a replacement hardware TOTP token for an IAM user, follow the steps in the procedure Enable a hardware TOTP token for another IAM user (console) earlier in this topic.

- To add a replacement hardware TOTP token for the AWS account root user, follow the steps in the procedure Enable a hardware TOTP token for the root user (console) earlier in this topic.

Assign MFA devices in the AWS CLI or AWS API

You can use AWS CLI commands or AWS API operations to enable a virtual MFA device for an IAM user. You cannot enable an MFA device for the AWS account root user with the AWS CLI, AWS API, Tools for Windows PowerShell, or any other command line tool. However, you can use the AWS Management Console to enable an MFA device for the root user.

When you enable an MFA device from the AWS Management Console, the console performs multiple steps for you. If you instead create a virtual device using the AWS CLI, Tools for Windows PowerShell, or AWS API, then you must perform the steps manually and in the correct order. For example, to create a virtual MFA device, you must create the IAM object and extract the code as either a string or a QR code graphic. Then you must sync the device and associate it with an IAM user. See the **Examples** section of New-IAMVirtualMFADevice for more details. For a physical device, you skip the creation step and go directly to syncing the device and associating it with the user.

You can attach tags to your IAM resources, including virtual MFA devices, to identify, organize, and control access to them. You can tag virtual MFA devices only when you use the AWS CLI or AWS API.

An IAM user using the SDK or CLI can enable an additional MFA device by calling `EnableMFADevice` or deactivate an existing MFA device by calling `DeactivateMFADevice`. To do this successfully, they must first call `GetSessionToken` and submit MFA codes with an existing MFA device. This call returns temporary security credentials that can then be used to sign API operations that require MFA authentication. For an example request and response, see `GetSessionToken`—temporary credentials for users in untrusted environments.

To create the virtual device entity in IAM to represent a virtual MFA device

These commands provide an ARN for the device that is used in place of a serial number in many of the following commands.

- AWS CLI: `aws iam create-virtual-mfa-device`
- AWS API: `CreateVirtualMFADevice`

To enable an MFA device for use with AWS

These commands synchronize the device with AWS and associate it with a user. If the device is virtual, use the ARN of the virtual device as the serial number.

> **Important**
>
> Submit your request immediately after generating the authentication codes. If you generate the codes and then wait too long to submit the request, the MFA device successfully associates with the user but the MFA device becomes out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resynchronize the device using the commands described below.

- AWS CLI: `aws iam enable-mfa-device`
- AWS API: `EnableMFADevice`

To deactivate a device

Use these commands to disassociate the device from the user and deactivate it. If the device is virtual, use the ARN of the virtual device as the serial number. You must also separately delete the virtual device entity.

- AWS CLI: `aws iam deactivate-mfa-device`
- AWS API: `DeactivateMFADevice`

To list virtual MFA device entities

Use these commands to list virtual MFA device entities.

- AWS CLI: `aws iam list-virtual-mfa-devices`
- AWS API: `ListVirtualMFADevices`

To tag a virtual MFA device

Use these commands to tag a virtual MFA device.

- AWS CLI: `aws iam tag-mfa-device`
- AWS API: `TagMFADevice`

To list tags for a virtual MFA device

Use these commands to list the tags attached to a virtual MFA device.

- AWS CLI: `aws iam list-mfa-device-tags`
- AWS API: `ListMFADeviceTags`

To untag a virtual MFA device

Use these commands to remove tags attached to a virtual MFA device.

- AWS CLI: `aws iam untag-mfa-device`
- AWS API: `UntagMFADevice`

To resynchronize an MFA device

Use these commands if the device is generating codes that are not accepted by AWS. If the device is virtual, use the ARN of the virtual device as the serial number.

- AWS CLI: `aws iam resync-mfa-device`
- AWS API: `ResyncMFADevice`

To delete a virtual MFA device entity in IAM

After the device is disassociated from the user, you can delete the device entity.

- AWS CLI: `aws iam delete-virtual-mfa-device`
- AWS API: `DeleteVirtualMFADevice`

To recover a virtual MFA device that is lost or not working

Sometimes, a user's device that hosts the virtual MFA app is lost, replaced, or not working. When this happens, the user can't recover it on their own. The user must contact an administrator to deactivate the device. For more information, see Recover an MFA protected identity in IAM.

Check MFA status

Use the IAM console to check whether an AWS account root user or IAM user has a valid MFA device enabled.

To check the MFA status of a root user

1. Sign in to the AWS Management Console with your root user credentials and then open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.
3. Check under **Multi-factor Authentication (MFA)** to see whether MFA is enabled or disabled. If MFA has not been activated, an alert symbol () is displayed.

If you want to enable MFA for the account, see one of the following:

- Enable a virtual MFA device for the root user (console)
- Enable a passkey or security key for the root user (console)
- Enable a hardware TOTP token for the root user (console)

To check the MFA status of IAM users

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. If necessary, add the **MFA** column to the users table by completing the following steps:
   a. Above the table on the far right, choose the settings icon ().
   b. In **Manage Columns**, select **MFA**.
   c. (Optional) Clear the checkbox for any column headings that you do not want to appear in the users table.
   d. Choose **Close** to return to the list of users.
4. The **MFA** column tells you about the MFA device that is enabled. If no MFA device is active for the user, the console displays **None**. If the user has an MFA device enabled, the **MFA** column shows the type of device that is enabled with a value of **Virtual**, **Security key**, **Hardware**, or **SMS**.

> **Note**
>
> AWS ended support for enabling SMS multi-factor authentication (MFA). We recommend that customers who have IAM users that use SMS text message-based MFA switch to one of the following alternative methods: virtual (software-based) MFA device, FIDO security key, or hardware MFA device. You can identify the users in your account with an assigned SMS MFA device. To do so, go to the IAM console, choose **Users** from the navigation pane, and look for users with **SMS** in the **MFA** column of the table.

5. To view additional information about the MFA device for a user, choose the name of the user whose MFA status you want to check. Then choose the **Security credentials** tab.
6. If no MFA device is active for the user, the console displays **No MFA devices. Assign an MFA device to improve the security of your AWS environment** in the **Multi-factor authentication (MFA)** section. If the user has MFA devices enabled, the **Multi-factor authentication (MFA)** section shows details about the devices:
   - The device name
   - The device type
   - The identifier for the device, such as a serial number for a physical device or the ARN in AWS for a virtual device
   - When the device was created

To remove or resync a device, choose the radio button next to the device and choose **Remove** or **Resync**.

For more information on enabling MFA, see the following:

- Assign a virtual MFA device in the AWS Management Console
- Assign a passkey or security key in the AWS Management Console
- Assign a hardware TOTP token in the AWS Management Console

Resynchronize virtual and hardware MFA devices

You can use AWS to resynchronize your virtual and hardware multi-factor authentication (MFA) devices. If your device is not synchronized when you try to use it, the sign-in attempt fails and IAM prompts you to resynchronize the device.

> **Note**
>
> FIDO security keys do not go out of sync. If a FIDO security key is lost or broken, you can deactivate it. For instructions on deactivating any MFA device type, see To deactivate an MFA device for another IAM user (console).

As an AWS administrator, you can resynchronize your IAM users' virtual and hardware MFA devices if they get out of synchronization.

If your AWS account root user MFA device is not working, you can resynchronize your device using the IAM console with or without completing the sign-in process. If you aren't able to successfully resynchronize your device, you may need to de-associate and re-associate it. For more information on how to do this, see Deactivate an MFA device and AWS Multi-factor authentication in IAM.

Topics

- Permissions required
- Resynchronizing virtual and hardware MFA devices (IAM console)
- Resynchronizing virtual and hardware MFA devices (AWS CLI)
- Resynchronizing virtual and hardware MFA devices (AWS API)

Permissions required

To resynchronize virtual or hardware MFA devices for your own IAM user, you must have the permissions from the following policy. This policy does not allow you to create or deactivate a device.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowListActions",
            "Effect": "Allow",
            "Action": [
                "iam:ListVirtualMFADevices"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowUserToViewAndManageTheirOwnUserMFA",
            "Effect": "Allow",
            "Action": [
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "BlockAllExceptListedIfNoMFA",
            "Effect": "Deny",
            "NotAction": [
                "iam:ListMFADevices",
                "iam:ListVirtualMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "*",
            "Condition": {
                "BoolIfExists": {
                    "aws:MultiFactorAuthPresent": "false"
                }
            }
        }
    ]
}
```

Resynchronizing virtual and hardware MFA devices (IAM console)

You can use the IAM console to resynchronize virtual and hardware MFA devices.

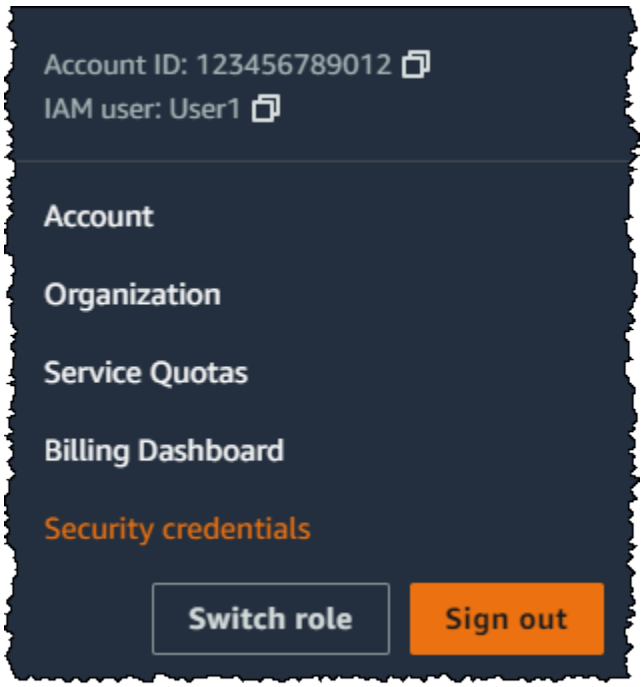
To resynchronize a virtual or hardware MFA device for your own IAM user (console)

1. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the IAM console.

   > **Note**
   >
   > For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose **Sign in to a different account** near the bottom of the page to return to the main sign-in page. From there, you can type your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

   To get your AWS account ID, contact your administrator.

2. In the navigation bar on the upper right, choose your user name, and then choose **Security credentials**.



3. On the **AWS IAM credentials** tab, in the **Multi-factor authentication (MFA)** section, choose the radio button next to the MFA device and choose **Resync**.
4. Type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Resync**.

> **Important**
>
> Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the request appears to work but the device remains out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time.

To resynchronize a virtual or hardware MFA device for another IAM user (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**, and then choose the name of the user whose MFA device needs to be resynchronized.
3. Choose the **Security credentials** tab. In the **Multi-factor authentication (MFA)** section, choose the radio button next to the MFA device and choose **Resync**.
4. Type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Resync**.

> **Important**
>
> Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the request appears to work but the device remains out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time.

To resynchronize your root user MFA before signing in (console)

1. On the **Amazon Web Services Sign In With Authentication Device** page, choose **Having problems with your authentication device? Click here**.

> **Note**
>
> You might see different text, such as **Sign in using MFA** and **Troubleshoot your authentication device**. However, the same features are provided.

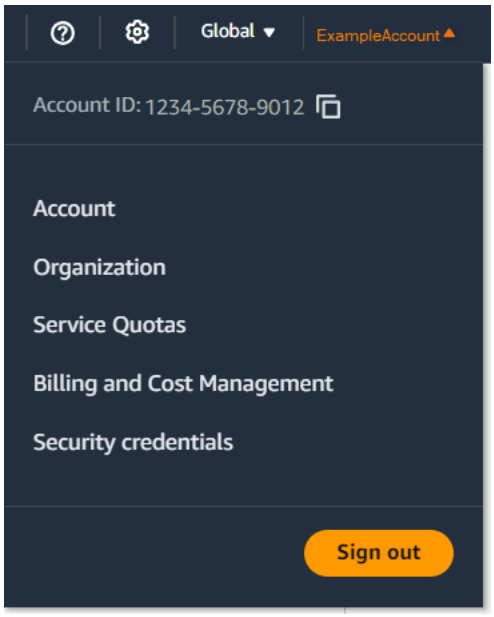
2. In the **Re-Sync With Our Servers** section, type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Re-sync authentication device**.
3. If necessary, type your password again and choose **Sign in**. Then complete the sign-in using your MFA device.

To resynchronize your root user MFA device after signing in (console)

1. Sign in to the IAM console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   > **Note**
   >
   > As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see Signing in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.

2. On the right side of the navigation bar, choose on your account name, and then choose **Security credentials**. If necessary, choose **Continue to Security credentials**.



3. Expand the **Multi-factor authentication (MFA)** section on the page.
4. Choose the radio button next to the device and choose **Resync**.
5. In the **Resync MFA device** dialog box, type the next two sequentially generated codes from the device into **MFA code 1** and **MFA code 2**. Then choose **Resync**.

> **Important**
>
> Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the MFA device is successfully associated with the user, but the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time.

Resynchronizing virtual and hardware MFA devices (AWS CLI)

You can resynchronize virtual and hardware MFA devices from the AWS CLI.

To resynchronize a virtual or hardware MFA device for an IAM user (AWS CLI)

At a command prompt, issue the aws iam resync-mfa-device command:

- Virtual MFA device: Specify Amazon Resource Name (ARN) of device as the serial number.

```
aws iam resync-mfa-device --user-name Richard --serial-number
 arn:aws:iam::123456789012:mfa/RichardsMFA --authentication-code1 123456 --
authentication-code2 987654
```

- Hardware MFA device: Specify hardware device's serial number as serial number. The format is vendor-specific. For example, you can purchase a gemalto token from Amazon. Its serial number is typically four letters followed by four numbers.

```
aws iam resync-mfa-device --user-name Richard --serial-number ABCD12345678 --
authentication-code1 123456 --authentication-code2 987654
```

> **Important**
>
> Submit your request immediately after generating the codes. If you generate the codes and then wait too long to submit the request, the request fails because the codes expire after a short time.

Resynchronizing virtual and hardware MFA devices (AWS API)

IAM has an API call that performs synchronization. In this case, we recommend that you give your virtual and hardware MFA device users permission to access this API call. Then build a tool based on that API call so your users can resynchronize their devices whenever they need to.

To resynchronize a virtual or hardware MFA device for an IAM user (AWS API)

- Send the ResyncMFADevice request.

Deactivate an MFA device

If you are having trouble signing in with a multi-factor authentication (MFA) device as an IAM user, contact your administrator for help.

As an administrator, you can deactivate the device for another IAM user. This allows the user to sign in without using MFA. You might do this as a temporary solution while the MFA device is replaced, or if the device is temporarily unavailable. However, we recommend that you enable a new device for the user as soon as possible. To learn how to enable a new MFA device, see AWS Multi-factor authentication in IAM.

> **Note**
>
> If you use the API or AWS CLI to delete a user from your AWS account, you must deactivate or delete the user's MFA device. You make this change as part of the process of removing the user. For more information about removing users, see Remove or deactivate an IAM user.

Topics

- Deactivating MFA devices (console)
- Deactivating MFA devices (AWS CLI)
- Deactivating MFA devices (AWS API)

Deactivating MFA devices (console)

To deactivate an MFA device for another IAM user (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. To deactivate the MFA device for a user, choose the name of the user whose MFA you want to remove.
4. Choose the **Security credentials** tab.
5. Under **Multi-factor authentication (MFA)**, choose the radio button next to the MFA device, choose **Remove**, and then choose **Remove**.

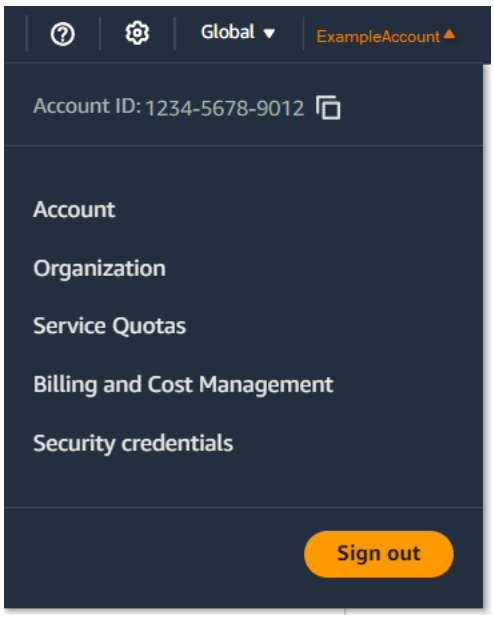
   The device is removed from AWS. It cannot be used to sign in or authenticate requests until it is reactivated and associated with an AWS user or AWS account root user.

To deactivate the MFA device for your AWS account root user (console)

1. Sign in to the IAM console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   > **Note**
   >
   > As the root user, you can't sign in to the **Sign in as IAM user** page. If you see the **Sign in as IAM user** page, choose **Sign in using root user email** near the bottom of the page. For help signing in as the root user, see Signing in to the AWS Management Console as the root user in the *AWS Sign-In User Guide*.

2. On the right side of the navigation bar, choose on your account name, and then choose **Security credentials**. If necessary, choose **Continue to Security credentials**.



3. In the **Multi-factor authentication (MFA)** section, choose the radio button next the MFA device that you want to deactivate and choose **Remove**.
4. Choose **Remove**.

   The MFA device is deactivated for the AWS account. Check the email that is associated with your AWS account for a confirmation message from Amazon Web Services. The email informs you that your Amazon Web Services multi-factor authentication (MFA) has been deactivated. The message will come from `@amazon.com` or `@aws.amazon.com`.

> **Note**
>
> Unassigned virtual MFA devices in your AWS account are deleted when you're adding new virtual MFA devices either via the AWS Management Console or during the sign-in process. Unassigned virtual MFA devices are devices in your account but not used by account root user or IAM users for the sign-in process. They're deleted so new virtual MFA devices can be added to your account. It also allows you to reuse device names.

Deactivating MFA devices (AWS CLI)

To deactivate an MFA device for an IAM user (AWS CLI)

- Run this command: `aws iam deactivate-mfa-device`

Deactivating MFA devices (AWS API)

To deactivate an MFA device for an IAM user (AWS API)

- Call this operation: `DeactivateMFADevice`

Recover an MFA protected identity in IAM

If your virtual MFA device or hardware TOTP token appears to be functioning properly, but you can't use it to access your AWS resources, it might be out of synchronization with AWS. For information about synchronizing a virtual MFA device or hardware MFA device, see Resynchronize virtual and hardware MFA devices. FIDO security keys do not go out of sync.

If the MFA device for a AWS account root user is lost, damaged, or not working, you can recover access to your account. IAM users must contact an administrator to deactivate the device.

> **Important**
>
> We recommend that you activate multiple MFA devices. Registering multiple MFA devices helps ensure continued access if a device is lost or broken. Your AWS account root user and IAM users can register up to eight MFA devices of any type.

Prerequisite – Use another MFA device

If your multi-factor authentication (MFA) device is lost, damaged, or not working, you can sign in using another MFA device registered to the same root user or IAM user.

To sign in using another MFA device

1. Sign in to the AWS Management Console with your AWS account ID or account alias and password.
2. On the **Additional verification required** page or **Multi-factor authentication** page, choose **Try another MFA method**.
3. Authenticate with the type of MFA device that you selected.
4. The next step varies based on whether you successfully signed in with an alternate MFA device.
   - If you have successfully signed in, you can Resynchronize virtual and hardware MFA devices, which may resolve the issue. If your MFA device is lost or broken, you can

deactivate it. For instructions on deactivating any MFA device type, see Deactivate an MFA device.

- If you can't sign in with MFA, use the steps in Recovering a root user MFA device or Recovering an IAM user MFA device to recover your MFA protected identity.

Recovering a root user MFA device

If you can't sign in with MFA, you can use alternative methods of authentication to sign in by verifying your identity using the email and the primary contact phone number registered with your account.

Confirm you are able to access the email and primary contact phone number associated with your account before you use alternative authentication factors to sign in as a root user. If you need to update the primary contact phone number, sign in as an IAM user with *Administrator* access instead of the root user. For additional instructions on updating the account contact information, see Editing contact information in the *AWS Billing User Guide*. If you do not have access to an email and primary contact phone number, you must contact AWS Support.

> **Important**
>
> We recommend that you keep the email address and contact phone number linked to your root user up to date for a successful account recovery. For more information, see Update the primary contact for your AWS account in the *AWS Account Management Reference Guide*.

To sign in using alternative factors of authentication as an AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.
2. On the **Additional verification required** page, select an MFA method to authenticate with and choose **Next**.

> **Note**
>
> You might see alternative text, such as **Sign in using MFA**, **Troubleshoot your authentication device**, or **Troubleshoot MFA**, but the functionality is the same. If you

can't use alternative authentication factors to verify your account email address and primary contact phone number, contact AWS Support to deactivate your MFA device.

3. Depending on the type of MFA you are using, you will see a different page, but the **Troubleshoot MFA** option functions the same. On the **Additional verification required** page or **Multi-factor authentication** page, choose **Troubleshoot MFA**.
4. If required, type your password again and choose **Sign in**.
5. On the **Troubleshoot your authentication device** page, in the **Sign in using alternative factors of authentication** section, choose **Sign in using alternative factors**.
6. On the **Sign in using alternative factors of authentication** page, authenticate your account by verifying the email address, choose **Send verification email**.
7. Check the email that is associated with your AWS account for a message from Amazon Web Services (recover-mfa-no-reply@verify.signin.aws). Follow the directions in the email.

   If you don't see the email in your account, check your spam folder, or return to your browser and choose **Resend the email**.
8. After you verify your email address, you can continue authenticating your account. To verify your primary contact phone number, choose **Call me now**.
9. Answer the call from AWS and, when prompted, enter the 6-digit number from the AWS website on your phone keypad.

   If you don't receive a call from AWS, choose **Sign in** to sign in to the console again and start over. Or see Lost or unusable Multi-Factor Authentication (MFA) device to contact support for help.
10. After you verify your phone number, you can sign in to your account by choosing **Sign in to the console**.
11. The next step varies depending on the type of MFA you are using:

    - For a virtual MFA device, remove the account from your device. Then go to the AWS Security Credentials page and delete the old MFA virtual device entity before you create a new one.
    - For a FIDO security key, see the AWS Security Credentials page and deactivate the old FIDO security key before enabling a new one.
    - For a hardware TOTP token, contact the third-party provider for help with fixing or replacing the device. You can continue to sign in using alternative factors of authentication until you receive your new device. After you have the new hardware MFA device, see the AWS Security Credentials page and delete the old MFA device.

> **Note**
>
> You don't have to replace a lost or stolen MFA device with the same type of device. For example, if you break your FIDO security key and order a new one, you can use virtual MFA or a hardware TOTP token until the new FIDO key arrives.

> **Important**
>
> If your MFA device is missing or stolen, change your root user password after signing in and establishing your replacement MFA device. An attacker may have stolen the authentication device and might also have your current password. For more information, see Change the password for the AWS account root user.

Recovering an IAM user MFA device

If you are an IAM user that can't sign in with MFA, you can't recover an MFA device by yourself. You must contact an administrator to deactivate the device. Then you can enable a new device.

To get help for an MFA device as an IAM user

1. Contact the AWS administrator or other person who gave you the user name and password for the IAM user. The administrator must deactivate the MFA device as described in Deactivate an MFA device so that you can sign in.
2. The next step varies depending on the type of MFA you are using:
   - For a virtual MFA device, remove the account from your device. Then enable the virtual device as described in Assign a virtual MFA device in the AWS Management Console.
   - For a FIDO security key, contact the third-party provider for help with replacing the device. When you receive the new FIDO security key, enable it as described in Assign a passkey or security key in the AWS Management Console.
   - For a hardware TOTP token, contact the third-party provider for help with fixing or replacing the device. After you have the new physical MFA device, enable the device as described in Assign a hardware TOTP token in the AWS Management Console.

> **Note**
>
> You don't have to replace a lost or stolen MFA device with the same type of device. You can have up to eight MFA devices of any combination. For example, if you break your FIDO security key and order a new one, you can use virtual MFA or a hardware TOTP token until the new FIDO key arrives.

3. If your MFA device is missing or stolen, also change your password in case an attacker has stolen the authentication device and might also have your current password. For more information, see Manage passwords for IAM users

Secure API access with MFA

With IAM policies, you can specify which API operations a user is allowed to call. You can apply additional security by requiring users to authenticate with multi-factor authentication (MFA) before you allow them to perform particularly sensitive actions.

For example, you might have a policy that allows a user to perform the Amazon EC2 `RunInstances`, `DescribeInstances`, and `StopInstances` actions. But you might want to restrict a destructive action like `TerminateInstances` and ensure that users can perform that action only if they authenticate with an AWS MFA device.

Topics

- Overview
- Scenario: MFA protection for cross-account delegation
- Scenario: MFA protection for access to API operations in the current account
- Scenario: MFA protection for resources that have resource-based policies

Overview

Adding MFA protection to API operations involves these tasks:

1. The administrator configures an AWS MFA device for each user who must make API requests that require MFA authentication. For more information, see AWS Multi-factor authentication in IAM.

2. The administrator creates policies for the users that include a `Condition` element that checks whether the user authenticated with an AWS MFA device.
3. The user calls one of the AWS STS API operations that support the MFA parameters: AssumeRole or GetSessionToken. As part of the call, the user includes the device identifier for the device that's associated with the user. The user also includes the time-based one-time password (TOTP) that the device generates. In either case, the user gets back temporary security credentials that the user can then use to make additional requests to AWS.

> **Note**
>
> MFA protection for a service's API operations is available only if the service supports temporary security credentials. For a list of these services, see Using Temporary Security Credentials to Access AWS.

If authorization fails, AWS returns an access denied error message (as it does for any unauthorized access). With MFA-protected API policies in place, AWS denies access to the API operations specified in the policies if the user attempts to call an API operation without valid MFA authentication. The operation is also denied if the time stamp of the request for the API operation is outside of the allowed range specified in the policy. The user must be reauthenticated with MFA by requesting new temporary security credentials with an MFA code and device serial number.

IAM policies with MFA conditions

Policies with MFA conditions can be attached to the following:

- An IAM user or group
- A resource such as an Amazon S3 bucket, Amazon SQS queue, or Amazon SNS topic
- The trust policy of an IAM role that can be assumed by a user

You can use an MFA condition in a policy to check the following properties:

- Existence—To simply verify that the user did authenticate with MFA, check that the `aws:MultiFactorAuthPresent` key is `True` in a `Bool` condition. The key is only present when the user authenticates with short-term credentials. Long-term credentials, such as access keys, do not include this key.

- Duration—If you want to grant access only within a specified time after MFA authentication, use a numeric condition type to compare the `aws:MultiFactorAuthAge` key's age to a value (such as 3600 seconds). Note that the `aws:MultiFactorAuthAge` key is not present if MFA was not used.

The following example shows the trust policy of an IAM role that includes an MFA condition to test for the existence of MFA authentication. With this policy, users from the AWS account specified in the `Principal` element (replace `ACCOUNT-B-ID` with a valid AWS account ID) can assume the role that this policy is attached to. However such users can only assume the role if the user is authenticated using MFA.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "ACCOUNT-B-ID"},
    "Action": "sts:AssumeRole",
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }
}
```

For more information on the condition types for MFA, see AWS global condition context keys, Numeric condition operators, and Condition operator to check existence of condition keys.

Choosing between GetSessionToken and AssumeRole

AWS STS provides two API operations that let users pass MFA information: `GetSessionToken` and `AssumeRole`. The API operation that the user calls to get temporary security credentials depends on which of the following scenarios applies.

Use `GetSessionToken` for the following scenarios:

- Call API operations that access resources in the same AWS account as the IAM user who makes the request. Note that temporary credentials from a `GetSessionToken` request can access IAM and AWS STS API operations *only* if you include MFA information in the request for credentials.

Because temporary credentials returned by `GetSessionToken` include MFA information, you can check for MFA in individual API operations made by the credentials.

- Access to resources that are protected with resource-based policies that include an MFA condition.

The purpose of the `GetSessionToken` operation is to authenticate the user using MFA. You cannot use policies to control authentication operations.

Use `AssumeRole` for the following scenarios:

- Call API operations that access resources in the same or a different AWS account. The API calls can include any IAM or AWS STS API. Note that to protect access you enforce MFA at the time when the user assumes the role. The temporary credentials returned by `AssumeRole` do not include MFA information in the context, so you cannot check individual API operations for MFA. This is why you must use `GetSessionToken` to restrict access to resources protected by resource-based policies.

> **Note**
>
> AWS CloudTrail logs will contain MFA information when the IAM user signs in with MFA. If the IAM user assumes an IAM role, CloudTrail will also log `mfaAuthenticated: true` in the `sessionContext` attributes for actions performed using the assumed role. However, CloudTrail logging is separate from what IAM requires when API calls are made with the assumed role's credentials. For more information, see CloudTrail userIdentity Element.

Details about how to implement these scenarios are provided later in this document.

Important points about MFA-protected API access

It's important to understand the following aspects of MFA protection for API operations:

- MFA protection is available only with temporary security credentials, which must be obtained with `AssumeRole` or `GetSessionToken`.
- You cannot use MFA-protected API access with AWS account root user credentials.
- You cannot use MFA-protected API access with U2F security keys.

- Federated users cannot be assigned an MFA device for use with AWS services, so they cannot access AWS resources controlled by MFA. (See next point.)
- Other AWS STS API operations that return temporary credentials do not support MFA. For `AssumeRoleWithWebIdentity` and `AssumeRoleWithSAML`, the user is authenticated by an external provider and AWS cannot determine whether that provider required MFA. For `GetFederationToken`, MFA is not necessarily associated with a specific user.
- Similarly, long-term credentials (IAM user access keys and root user access keys) cannot be used with MFA-protected API access because they don't expire.
- `AssumeRole` and `GetSessionToken` can also be called without MFA information. In that case, the caller gets back temporary security credentials, but the session information for those temporary credentials does not indicate that the user authenticated with MFA.
- To establish MFA protection for API operations, you add MFA conditions to policies. A policy must include the `aws:MultiFactorAuthPresent` condition key to enforce the use of MFA. For cross-account delegation, the role's trust policy must include the condition key.
- When you allow another AWS account to access resources in your account, the security of your resources depends on the configuration of the trusted account (the other account, not yours). This is true even when you require multi-factor authentication. Any identity in the trusted account that has permission to create virtual MFA devices can construct an MFA claim to satisfy that part of your role's trust policy. Before you allow members of another account access to your AWS resources that require multi-factor authentication, you should ensure that the trusted account's owner follows security best practices. For example, the trusted account should restrict access to sensitive API operations, such as MFA device-management API operations, to specific, trusted identities.
- If a policy includes an MFA condition, a request is denied if users have not been MFA authenticated, or if they provide an invalid MFA device identifier or invalid TOTP.

Scenario: MFA protection for cross-account delegation

In this scenario, you want to delegate access to IAM users in another account, but only if the users are authenticated with an AWS MFA device. For more information about cross-account delegation, see Roles terms and concepts.

Imagine that you have account A (the trusting account that owns the resource to be accessed), with the IAM user Anaya, who has administrator permission. She wants to grant access to user Richard in account B (the trusted account), but wants to make sure that Richard is authenticated with MFA before he assumes the role.

1. In the trusting account A, Anaya creates an IAM role named `CrossAccountRole` and sets the principal in the role's trust policy to the account ID of account B. The trust policy grants permission to the AWS STS `AssumeRole` action. Anaya also adds an MFA condition to the trust policy, as in the following example.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": {
       "Effect": "Allow",
       "Principal": {"AWS": "ACCOUNT-B-ID"},
       "Action": "sts:AssumeRole",
       "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
     }
   }
   ```

2. Anaya adds a permissions policy to the role that specifies what the role is allowed to do. The permissions policy for a role with MFA protection is no different than any other role-permission policy. The following example shows the policy that Anaya adds to the role; it allows an assuming user to perform any Amazon DynamoDB action on the table `Books` in account A. This policy also allows the `dynamodb:ListTables` action, which is required to perform actions in the console.

   > **Note**
   >
   > The permissions policy does not include an MFA condition. It is important to understand that the MFA authentication is used only to determine whether a user can assume the role. Once the user has assumed the role, no further MFA checks are made.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "TableActions",
               "Effect": "Allow",
               "Action": "dynamodb:*",
   ```

```
            "Resource": "arn:aws:dynamodb:*:111122223333:table/Books"
        },
        {
            "Sid": "ListTables",
            "Effect": "Allow",
            "Action": "dynamodb:ListTables",
            "Resource": "*"
        }
    ]
}
```

3. In trusted account B, the administrator makes sure that IAM user Richard is configured with an AWS MFA device and that he knows the ID of the device. The device ID is the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
4. In account B, the administrator attaches the following policy to user Richard (or a group that he's a member of) that allows him to call the `AssumeRole` action. The resource is set to the ARN of the role that Anaya created in step 1. Notice that this policy does not contain an MFA condition.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::111122223333:role/CrossAccountRole"
            ]
        }
    ]
}
```

5. In account B, Richard (or an application that Richard is running) calls `AssumeRole`. The API call includes the ARN of the role to assume (`arn:aws:iam::`*`ACCOUNT-A-ID`*`:role/CrossAccountRole`), the ID of the MFA device, and the current TOTP that Richard gets from his device.

   When Richard calls `AssumeRole`, AWS determines whether he has valid credentials, including the requirement for MFA. If so, Richard successfully assumes the role and can perform any

DynamoDB action on the table named `Books` in account A while using the role's temporary credentials.

For an example of a program that calls `AssumeRole`, see Calling AssumeRole with MFA authentication.

Scenario: MFA protection for access to API operations in the current account

In this scenario, you should ensure that a user in your AWS account can access sensitive API operations only when the user is authenticated using an AWS MFA device.

Imagine that you have account A that contains a group of developers who need to work with EC2 instances. Ordinary developers can work with the instances, but they are not granted permissions for the `ec2:StopInstances` or `ec2:TerminateInstances` actions. You want to limit those "destructive" privileged actions to just a few trusted users, so you add MFA protection to the policy that allows these sensitive Amazon EC2 actions.

In this scenario, one of those trusted users is user Sofía. User Anaya is an administrator in account A.

1. Anaya makes sure that Sofía is configured with an AWS MFA device and that Sofía knows the ID of the device. The device ID is the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
2. Anaya creates a group named `EC2-Admins` and adds user Sofía to the group.
3. Anaya attaches the following policy to the `EC2-Admins` group. This policy grants users permission to call the Amazon EC2 `StopInstances` and `TerminateInstances` actions only if the user has authenticated using MFA.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "ec2:StopInstances",
      "ec2:TerminateInstances"
    ],
    "Resource": ["*"],
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
```

```
  }]
}
```

4. **Note**

   For this policy to take effect, users must first sign out and then sign in again.

   If user Sofía needs to stop or terminate an Amazon EC2 instance, she (or an application that she is running) calls `GetSessionToken`. This API operation passes the ID of the MFA device and the current TOTP that Sofía gets from her device.

5. User Sofía (or an application that Sofía is using) uses the temporary credentials provided by `GetSessionToken` to call the Amazon EC2 `StopInstances` or `TerminateInstances` action.

   For an example of a program that calls `GetSessionToken`, see Calling GetSessionToken with MFA authentication later in this document.

Scenario: MFA protection for resources that have resource-based policies

In this scenario, you are the owner of an S3 bucket, an SQS queue, or an SNS topic. You want to make sure that any user from any AWS account who accesses the resource is authenticated by an AWS MFA device.

This scenario illustrates a way to provide cross-account MFA protection without requiring users to assume a role first. In this case, the user can access the resource if three conditions are met: The user must be authenticated by MFA, be able to get temporary security credentials from `GetSessionToken`, and be in an account that is trusted by the resource's policy.

Imagine that you are in account A and you create an S3 bucket. You want to grant access to this bucket to users who are in several different AWS accounts, but only if those users are authenticated with MFA.

In this scenario, user Anaya is an administrator in account A. User Nikhil is an IAM user in account C.

1. In account A, Anaya creates a bucket named `Account-A-bucket`.
2. Anaya adds the bucket policy to the bucket. The policy allows any user in account A, account B, or account C to perform the Amazon S3 `PutObject` and `DeleteObject` actions in the bucket. The policy includes an MFA condition.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": [
      "ACCOUNT-A-ID",
      "ACCOUNT-B-ID",
      "ACCOUNT-C-ID"
    ]},
    "Action": [
      "s3:PutObject",
      "s3:DeleteObject"
    ],
    "Resource": ["arn:aws:s3:::ACCOUNT-A-BUCKET-NAME/*"],
    "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
  }]
}
```

> **Note**
>
> Amazon S3 offers an MFA Delete feature for *root* account access (only). You can enable Amazon S3 MFA Delete when you set the versioning state of the bucket. Amazon S3 MFA Delete cannot be applied to an IAM user, and is managed independently from MFA-protected API access. An IAM user with permissions to delete a bucket cannot delete a bucket with Amazon S3 MFA Delete enabled. For more information on Amazon S3 MFA Delete, see MFA Delete.

3. In account C, an administrator makes sure that user Nikhil is configured with an AWS MFA device and that he knows the ID of the device. The device ID is the serial number if it's a hardware MFA device, or the device's ARN if it's a virtual MFA device.
4. In account C, Nikhil (or an application that he is running) calls `GetSessionToken`. The call includes the ID or ARN of the MFA device and the current TOTP that Nikhil gets from his device.
5. Nikhil (or an application that he is using) uses the temporary credentials returned by `GetSessionToken` to call the Amazon S3 `PutObject` action to upload a file to `Account-A-bucket`.

For an example of a program that calls `GetSessionToken`, see Calling GetSessionToken with MFA authentication later in this document.

> **Note**
>
> The temporary credentials that `AssumeRole` returns won't work in this case. Although the user can provide MFA information to assume a role, the temporary credentials returned by `AssumeRole` don't include the MFA information. That information is required in order to meet the MFA condition in the policy.

Sample code: Requesting credentials with multi-factor authentication

The following examples show how to call `GetSessionToken` and `AssumeRole` operations and pass MFA authentication parameters. No permissions are required to call `GetSessionToken`, but you must have a policy that allows you to call `AssumeRole`. The credentials returned are then used to list all S3 buckets in the account.

Calling GetSessionToken with MFA authentication

The following example shows how to call `GetSessionToken` and pass MFA authentication information. The temporary security credentials returned by the `GetSessionToken` operation are then used to list all S3 buckets in the account.

The policy attached to the user who runs this code (or to a group that the user is in) provides the permissions for the returned temporary credentials. For this example code, the policy must grant the user permission to request the Amazon S3 `ListBuckets` operation.

The following code examples show how to use `GetSessionToken`.

CLI

AWS CLI

To get a set of short term credentials for an IAM identity

The following `get-session-token` example retrieves a set of short-term credentials for the IAM identity making the call. The resulting credentials can be used for requests where multi-factor authentication (MFA) is required by policy. The credentials expire 15 minutes after they are generated.

```
aws sts get-session-token \
    --duration-seconds 900 \
    --serial-number "YourMFADeviceSerialNumber" \
    --token-code 123456
```

Output:

```
{
    "Credentials": {
        "AccessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY",
        "SessionToken": "AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4OlgkBN9bkUDNCJiBeb/
AXlzBBko7b15fjrBs2+cTQtpZ3CYWFXG8C5zqx37wnOE49mRl/+OtkIKGO7fAE",
        "Expiration": "2020-05-19T18:06:10+00:00"
    }
}
```

For more information, see Requesting Temporary Security Credentials in the *AWS IAM User Guide*.

- For API details, see GetSessionToken in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for a set period of time. The credentials used to request temporary credentials are inferred from the current shell defaults. To specify other credentials, use the -ProfileName or -AccessKey/-SecretKey parameters.

```
Get-STSSessionToken
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
```

```
EXAMPLEACCESSKEYID                      2/16/2015 9:12:28 PM
 examplesecretaccesskey...              SamPleTokeN.....
```

Example 2: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour. The credentials used to make the request are obtained from the specified profile.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile
```

Output:

```
AccessKeyId                             Expiration
 SecretAccessKey                        SessionToken
-----------                             ----------
 ---------------                        ------------
EXAMPLEACCESSKEYID                      2/16/2015 9:12:28 PM
 examplesecretaccesskey...              SamPleTokeN.....
```

Example 3: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour using the identification number of the MFA device associated with the account whose credentials are specified in the profile 'myprofilename' and the value provided by the device.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile -SerialNumber
 YourMFADeviceSerialNumber -TokenCode 123456
```

Output:

```
AccessKeyId                             Expiration
 SecretAccessKey                        SessionToken
-----------                             ----------
 ---------------                        ------------
EXAMPLEACCESSKEYID                      2/16/2015 9:12:28 PM
 examplesecretaccesskey...              SamPleTokeN.....
```

- For API details, see GetSessionToken in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for a set period of time. The credentials used to request

temporary credentials are inferred from the current shell defaults. To specify other credentials, use the -ProfileName or -AccessKey/-SecretKey parameters.

```
Get-STSSessionToken
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

Example 2: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour. The credentials used to make the request are obtained from the specified profile.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

Example 3: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour using the identification number of the MFA device associated with the account whose credentials are specified in the profile 'myprofilename' and the value provided by the device.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile -SerialNumber
 YourMFADeviceSerialNumber -TokenCode 123456
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

- For API details, see GetSessionToken in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get a session token by passing an MFA token and use it to list Amazon S3 buckets for the account.

```
def list_buckets_with_session_token_with_mfa(mfa_serial_number, mfa_totp,
 sts_client):
    """
    Gets a session token with MFA credentials and uses the temporary session
    credentials to list Amazon S3 buckets.

    Requires an MFA device serial number and token.

    :param mfa_serial_number: The serial number of the MFA device. For a virtual
 MFA
                              device, this is an Amazon Resource Name (ARN).
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the
 role.
    """
    if mfa_serial_number is not None:
        response = sts_client.get_session_token(
            SerialNumber=mfa_serial_number, TokenCode=mfa_totp
        )
```

```
    else:
        response = sts_client.get_session_token()
    temp_credentials = response["Credentials"]

    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )

    print(f"Buckets for the account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
```

- For API details, see GetSessionToken in *AWS SDK for Python (Boto3) API Reference*.

Calling AssumeRole with MFA authentication

The following examples show how to call `AssumeRole` and pass MFA authentication information. The temporary security credentials returned by `AssumeRole` are then used to list all Amazon S3 buckets in the account.

For more information about this scenario, see Scenario: MFA protection for cross-account delegation.

The following code examples show how to use `AssumeRole`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace AssumeRoleExample
{
    class AssumeRole
    {
        /// <summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html
        /// for help in working with roles.
        /// </summary>

        // A region property may be used if the profile or credentials loaded do
 not specify a region,
        // or to use a specific region.
        private static readonly RegionEndpoint REGION = RegionEndpoint.USWest2;

        static async Task Main()
        {
            // Create the SecurityToken client and then display the identity of
 the
            // default user.
            var roleArnToAssume = "arn:aws:iam::123456789012:role/
testAssumeRole";

            var client = new
 Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

            // Get and display the information about the identity of the default
 user.
```

```
            var callerIdRequest = new GetCallerIdentityRequest();
            var caller = await client.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"Original Caller: {caller.Arn}");

            // Create the request to use with the AssumeRoleAsync call.
            var assumeRoleReq = new AssumeRoleRequest()
            {
                DurationSeconds = 1600,
                RoleSessionName = "Session1",
                RoleArn = roleArnToAssume
            };

            var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

            // Now create a new client based on the credentials of the caller
 assuming the role.
            var client2 = new AmazonSecurityTokenServiceClient(credentials:
 assumeRoleRes.Credentials, REGION);

            // Get and display information about the caller that has assumed the
 defined role.
            var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"AssumedRole Caller: {caller2.Arn}");
        }
    }
}
```

- For API details, see AssumeRole in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function iecho
```

```
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
##############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function sts_assume_role
#
# This function assumes a role in the AWS account and returns the temporary
#  credentials.
#
# Parameters:
#       -n role_session_name -- The name of the session.
#       -r role_arn -- The ARN of the role to assume.
#
# Returns:
#       [access_key_id, secret_access_key, session_token]
#     And:
#       0 - If successful.
#       1 - If an error occurred.
##############################################################################
function sts_assume_role() {
  local role_session_name role_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function sts_assume_role"
    echo "Assumes a role in the AWS account and returns the temporary
 credentials:"
```

```
    echo "  -n role_session_name -- The name of the session."
    echo "  -r role_arn -- The ARN of the role to assume."
    echo ""
  }

  while getopts n:r:h option; do
    case "${option}" in
      n) role_session_name=${OPTARG} ;;
      r) role_arn=${OPTARG} ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done

  response=$(aws sts assume-role \
    --role-session-name "$role_session_name" \
    --role-arn "$role_arn" \
    --output text \
    --query "Credentials.[AccessKeyId, SecretAccessKey, SessionToken]")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- For API details, see AssumeRole in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::STS::assumeRole(const Aws::String &roleArn,
                             const Aws::String &roleSessionName,
                             const Aws::String &externalId,
                             Aws::Auth::AWSCredentials &credentials,
                             const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::STS::STSClient sts(clientConfig);
    Aws::STS::Model::AssumeRoleRequest sts_req;

    sts_req.SetRoleArn(roleArn);
    sts_req.SetRoleSessionName(roleSessionName);
    sts_req.SetExternalId(externalId);

    const Aws::STS::Model::AssumeRoleOutcome outcome = sts.AssumeRole(sts_req);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error assuming IAM role. " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Credentials successfully retrieved." << std::endl;
        const Aws::STS::Model::AssumeRoleResult result = outcome.GetResult();
        const Aws::STS::Model::Credentials &temp_credentials =
 result.GetCredentials();

        // Store temporary credentials in return argument.
        // Note: The credentials object returned by assumeRole differs
        // from the AWSCredentials object used in most situations.
        credentials.SetAWSAccessKeyId(temp_credentials.GetAccessKeyId());
        credentials.SetAWSSecretKey(temp_credentials.GetSecretAccessKey());
        credentials.SetSessionToken(temp_credentials.GetSessionToken());
    }
```

```
    return outcome.IsSuccess();
}
```

- For API details, see AssumeRole in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To assume a role

The following `assume-role` example retrieves a set of short-term credentials for the IAM role `s3-access-example`.

```
aws sts assume-role \
    --role-arn arn:aws:iam::123456789012:role/xaccounts3access \
    --role-session-name s3-access-example
```

Output:

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA3XFRBF535PLBIFPI4:s3-access-example",
        "Arn": "arn:aws:sts::123456789012:assumed-role/xaccounts3access/s3-
access-example"
    },
    "Credentials": {
        "SecretAccessKey": "9drTJvcXLB89EXAMPLELB8923FB892xMFI",
        "SessionToken": "AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/
qwjzP2iEXAMPLEbw/m3hsj8VBTkPORGvr9jM5sgP+w9IZWZnU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYlFVgAUj
+7Indz3LU0aTWk1WKIjHmmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nyzM
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFiPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8B
IcrxSpnWEXAMPLEXSDFTAQAM6Dl9zR0tXoybnlrZIwMLlMi1Kcgo5OytwU=",
        "Expiration": "2016-03-15T00:05:07Z",
        "AccessKeyId": "ASIAJEXAMPLEXEG2JICEA"
    }
}
```

The output of the command contains an access key, secret key, and session token that you can use to authenticate to AWS.

For AWS CLI use, you can set up a named profile associated with a role. When you use the profile, the AWS CLI will call assume-role and manage credentials for you. For more information, see Use an IAM role in the AWS CLI in the *AWS CLI User Guide*.

- For API details, see AssumeRole in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sts.StsClient;
import software.amazon.awssdk.services.sts.model.AssumeRoleRequest;
import software.amazon.awssdk.services.sts.model.StsException;
import software.amazon.awssdk.services.sts.model.AssumeRoleResponse;
import software.amazon.awssdk.services.sts.model.Credentials;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * To make this code example work, create a Role that you want to assume.
 * Then define a Trust Relationship in the AWS Console. You can use this as an
 * example:
 *
 * {
 * "Version":"2012-10-17",
 * "Statement": [
 * {
 * "Effect": "Allow",
 * "Principal": {
```

```
 * "AWS": "<Specify the ARN of your IAM user you are using in this code example>"
 * },
 * "Action": "sts:AssumeRole"
 * }
 * ]
 * }
 *
 * For more information, see "Editing the Trust Relationship for an Existing
 * Role" in the AWS Directory Service guide.
 *
 * Also, set up your development environment, including your credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AssumeRole {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <roleArn> <roleSessionName>\s

                Where:
                    roleArn - The Amazon Resource Name (ARN) of the role to
 assume (for example, arn:aws:iam::000008047983:role/s3role).\s
                    roleSessionName - An identifier for the assumed role session
 (for example, mysession).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String roleArn = args[0];
        String roleSessionName = args[1];
        Region region = Region.US_EAST_1;
        StsClient stsClient = StsClient.builder()
                .region(region)
                .build();

        assumeGivenRole(stsClient, roleArn, roleSessionName);
```

```
        stsClient.close();
    }

    public static void assumeGivenRole(StsClient stsClient, String roleArn,
 String roleSessionName) {
        try {
            AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
                    .roleArn(roleArn)
                    .roleSessionName(roleSessionName)
                    .build();

            AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
            Credentials myCreds = roleResponse.credentials();

            // Display the time when the temp creds expire.
            Instant exTime = myCreds.expiration();
            String tokenInfo = myCreds.sessionToken();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
 DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.US)
                    .withZone(ZoneId.systemDefault());

            formatter.format(exTime);
            System.out.println("The token " + tokenInfo + "  expires on " +
 exTime);

        } catch (StsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see AssumeRole in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client.

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Assume the IAM role.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
```

```
  } catch (err) {
    console.error(err);
  }
};
```

- For API details, see AssumeRole in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
  DurationSeconds: 900,
};
var roleCreds;

// Create the STS service object
var sts = new AWS.STS({ apiVersion: "2011-06-15" });

//Assume Role
sts.assumeRole(roleToAssume, function (err, data) {
  if (err) console.log(err, err.stack);
  else {
    roleCreds = {
      accessKeyId: data.Credentials.AccessKeyId,
      secretAccessKey: data.Credentials.SecretAccessKey,
      sessionToken: data.Credentials.SessionToken,
    };
    stsGetCallerIdentity(roleCreds);
  }
});
```

```
//Get Arn of current identity
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);

  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- For API details, see AssumeRole in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns a set of temporary credentials (access key, secret key and session token) that can be used for one hour to access AWS resources that the requesting user might not normally have access to. The returned credentials have the permissions that are allowed by the access policy of the role being assumed and the policy that was supplied (you cannot use the supplied policy to grant permissions in excess of those defined by the access policy of the role being assumed).

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -Policy "...JSON policy..." -DurationInSeconds 3600
```

Example 2: Returns a set of temporary credentials, valid for one hour, that have the same permissions that are defined in the access policy of the role being assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600
```

Example 3: Returns a set of temporary credentials supplying the serial number and generated token from an MFA associated with the user credentials used to execute the cmdlet.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -SerialNumber "GAHT12345678" -TokenCode "123456"
```

Example 4: Returns a set of temporary credentials that have assumed a role defined in a customer account. For each role that the third party can assume, the customer account must create a role using an identifier that must be passed in the -ExternalId parameter each time the role is assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -ExternalId "ABC123"
```

- For API details, see AssumeRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns a set of temporary credentials (access key, secret key and session token) that can be used for one hour to access AWS resources that the requesting user might not normally have access to. The returned credentials have the permissions that are allowed by the access policy of the role being assumed and the policy that was supplied (you cannot use the supplied policy to grant permissions in excess of those defined by the access policy of the role being assumed).

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -Policy "...JSON policy..." -DurationInSeconds 3600
```

Example 2: Returns a set of temporary credentials, valid for one hour, that have the same permissions that are defined in the access policy of the role being assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600
```

Example 3: Returns a set of temporary credentials supplying the serial number and generated token from an MFA associated with the user credentials used to execute the cmdlet.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -SerialNumber "GAHT12345678" -TokenCode "123456"
```

Example 4: Returns a set of temporary credentials that have assumed a role defined in a customer account. For each role that the third party can assume, the customer account must create a role using an identifier that must be passed in the -ExternalId parameter each time the role is assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -ExternalId "ABC123"
```

- For API details, see AssumeRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Assume an IAM role that requires an MFA token and use temporary credentials to list Amazon S3 buckets for the account.

```
def list_buckets_from_assumed_role_with_mfa(
    assume_role_arn, session_name, mfa_serial_number, mfa_totp, sts_client
):
    """
    Assumes a role from another account and uses the temporary credentials from
    that role to list the Amazon S3 buckets that are owned by the other account.
    Requires an MFA device serial number and token.

    The assumed role must grant permission to list the buckets in the other
 account.

    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                            grants access to list the other account's buckets.
```

```
    :param session_name: The name of the STS session.
    :param mfa_serial_number: The serial number of the MFA device. For a virtual
MFA
                              device, this is an ARN.
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the
role.
    """
    response = sts_client.assume_role(
        RoleArn=assume_role_arn,
        RoleSessionName=session_name,
        SerialNumber=mfa_serial_number,
        TokenCode=mfa_totp,
    )
    temp_credentials = response["Credentials"]
    print(f"Assumed role {assume_role_arn} and got temporary credentials.")

    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )

    print(f"Listing buckets for the assumed role's account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
```

- For API details, see AssumeRole in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
 # Creates an AWS Security Token Service (AWS STS) client with specified
credentials.
 # This is separated into a factory function so that it can be mocked for unit
testing.
 #
 # @param key_id [String] The ID of the access key used by the STS client.
 # @param key_secret [String] The secret part of the access key used by the STS
client.
 def create_sts_client(key_id, key_secret)
   Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)
 end

 # Gets temporary credentials that can be used to assume a role.
 #
 # @param role_arn [String] The ARN of the role that is assumed when these
credentials
 #                          are used.
 # @param sts_client [AWS::STS::Client] An AWS STS client.
 # @return [Aws::AssumeRoleCredentials] The credentials that can be used to
assume the role.
 def assume_role(role_arn, sts_client)
   credentials = Aws::AssumeRoleCredentials.new(
     client: sts_client,
     role_arn: role_arn,
     role_session_name: 'create-use-assume-role-scenario'
   )
   @logger.info("Assumed role '#{role_arn}', got temporary credentials.")
   credentials
 end
```

- For API details, see AssumeRole in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn assume_role(config: &SdkConfig, role_name: String, session_name:
 Option<String>) {
    let provider = aws_config::sts::AssumeRoleProvider::builder(role_name)
        .session_name(session_name.unwrap_or("rust_sdk_example_session".into()))
        .configure(config)
        .build()
        .await;

    let local_config = aws_config::from_env()
        .credentials_provider(provider)
        .load()
        .await;
    let client = Client::new(&local_config);
    let req = client.get_caller_identity();
    let resp = req.send().await;
    match resp {
        Ok(e) => {
            println!("UserID :               {}",
 e.user_id().unwrap_or_default());
            println!("Account:               {}",
 e.account().unwrap_or_default());
            println!("Arn    :               {}", e.arn().unwrap_or_default());
        }
        Err(e) => println!("{:?}", e),
    }
}
```

- For API details, see AssumeRole in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSTS
```

```
public func assumeRole(role: IAMClientTypes.Role, sessionName: String)
    async throws -> STSClientTypes.Credentials
{
    let input = AssumeRoleInput(
        roleArn: role.arn,
        roleSessionName: sessionName
    )
    do {
        let output = try await stsClient.assumeRole(input: input)

        guard let credentials = output.credentials else {
            throw ServiceHandlerError.authError
        }

        return credentials
    } catch {
        print("Error assuming role: ", dump(error))
        throw error
    }
}
```

- For API details, see AssumeRole in *AWS SDK for Swift API reference*.

Service-specific credentials for IAM users

Service-specific credentials are specialized authentication mechanisms designed for specific AWS services. These credentials provide simplified authentication compared to standard AWS credentials, and are tailored to the authentication requirements of individual AWS services. Unlike access keys, which can be used across multiple AWS services, service-specific credentials are designed for use with only the service for which they were created. This targeted approach enhances security by limiting the scope of the credentials.

Service-specific credentials typically consist of a user name and password pair or specialized API keys that are formatted according to the requirements of the specific service. When you create service-specific credentials, they are active by default and can be used immediately. You can have a maximum of two sets of service-specific credentials for each supported service per IAM user. This limit allows you to maintain one active set while rotating to a new set when needed. AWS currently supports service-specific credentials for the following services:

When to use service-specific credentials

Service-specific credentials are intended for compatibility with third-party libraries, SDKs, tools, or applications that are not natively compatible with AWS credentials, AWS SDKs, or AWS APIs. Such use cases include migrating to AWS services from self-hosted infrastructure or from services hosted by other providers.

When starting from scratch, and wherever possible, we recommend that you use AWS temporary credentials, such as those provided by an IAM role, to authenticate with an AWS service using an AWS SDK or a library that supports AWS temporary credentials.

Rotating service-specific credentials

As a security best practice, rotate service-specific credentials regularly. To rotate credentials without disrupting your applications:

1. Create a second set of service-specific credentials for the same service and IAM user
2. Update all applications to use the new credentials and verify they work correctly
3. Change the status of the original credentials to "Inactive"
4. Verify that all applications are still functioning properly
5. Delete the inactive service-specific credentials when you're confident they're no longer needed

Monitoring service-specific credentials

You can use AWS CloudTrail to monitor the use of service-specific credentials in your AWS account. To view CloudTrail events related to service-specific credential usage, review the CloudTrail logs for events from the service where the credentials are used. For more information, see Logging IAM and AWS STS API calls with AWS CloudTrail.

For additional security, consider setting up CloudWatch alarms to notify you of specific credential usage patterns that might indicate unauthorized access or other security concerns. For more information, see Monitoring CloudTrail Log Files with Amazon CloudWatch Logs in the *AWS CloudTrail User Guide*.

The following topics provide information about service-specific credentials.

Topics

- API keys for AWS services

- Use IAM with Amazon Keyspaces (for Apache Cassandra)

API keys for AWS services

Some AWS services support API keys for authenticating programmatic requests in addition to standard IAM credentials such as temporary security credentials and long-term access keys. AWS offers two types of API keys:

- **Long-term API keys** – Long-term API keys are associated with an IAM user and generated using IAM service-specific credentials. These credentials are designed for use with only a single AWS service, enhancing security by limiting credential scope. You can set an expiration time for when the long-term API key expires. To generate long-term API keys, you can use the IAM or service-specific console, the AWS CLI, or AWS API.
- **Short-term API keys** – A short-term API key is a pre-signed URL that uses AWS Signature Version 4. Short-term API keys share the same permissions and expiration as the credentials of the identity that generates the API key and are valid for up to 12 hours or the remaining time of your console session, whichever is shorter. You can use the Amazon Bedrock/Claude Platform on AWS console, Python, and packages for other programming languages to generate short-term API keys. For more information, see Generate Amazon Bedrock API keys for easy access to the Amazon Bedrock API in the *Amazon Bedrock User Guide* and Authentication in the *Claude Platform on AWS User Guide*.

> **Note**
>
> Long-term API keys have a higher security risk compared to short-term API keys. We recommend using short-term API keys or temporary security credentials when possible. If you use long-term API keys, we recommend implementing regular key rotation practices.

Services that support API keys

The following table lists the AWS services that support API keys and the type of API key each service supports.

#	Service	Long-term API keys	Short-term API keys	Managed policy auto-attached	Service-specific documentation
1	Amazon Bedrock	Yes	Yes	AmazonBedrockLimitedAccess	Use an Amazon Bedrock API key
2	Claude Platform on AWS	Yes	Yes	AnthropicInferenceAccess	Authentication
3	Amazon CloudWatch	Yes	N/A	CloudWatchAPIKeyAccess	Setting up bearer token authentication for Metrics
4	Amazon CloudWatch Logs	Yes	N/A	CloudWatchLogsAPIKeyAccess	Setting up bearer token authentication

When you generate a long-term API key for a service, the corresponding AWS managed policy is automatically attached to the IAM user, granting access to core operations for that service. If you require additional access, you can modify the permissions for the IAM user. For information about modifying permissions, see the section called “Add or remove identity permissions”.

To learn more about API keys for specific services, refer to the Service-specific documentation links in the table above.

Prerequisites for long-term API keys

Before you can generate a long-term API key in the IAM console, you must meet these prerequisites:

- An IAM user to associate with the long-term API key. For instructions on creating an IAM user, see the section called “Create a user”.
- You must have the following IAM policy permissions to manage service-specific credentials for an IAM user. The example policy grants permission to create, list, update, delete, and reset service-specific credentials. Replace the *username* value in the Resource element with the name of the IAM user you will generate long-term API keys for:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageBedrockServiceSpecificCredentials",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceSpecificCredential",
                "iam:ListServiceSpecificCredentials",
                "iam:UpdateServiceSpecificCredential",
                "iam:DeleteServiceSpecificCredential",
                "iam:ResetServiceSpecificCredential"
            ],
            "Resource": "arn:aws:iam::*:user/username"
        }
    ]
}
```

Generating a long-term API key (console)

To generate a long-term API key for a specific service in the IAM console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Users**.
3. Choose the IAM user you want to generate a long-term API key for.
4. Choose the **Security credentials** tab.
5. In the **API keys** section, choose **Generate API key**.
6. From the **AWS service** dropdown list, choose the service that you want the API key to authenticate to.

7. For **API key expiration**, do one of the following:
   - Choose an API key expiration duration of **1**, **5**, **30**, **90**, or **365** days.
   - Choose **Custom duration** to specify a custom API key expiration date.
   - Choose **Never expires** (not recommended).
8. Choose **Generate API key**.
9. Copy or download your API key. This is the only time you can view the API key value.

> **Important**
>
> Store your API key securely. After you close the dialog box, you cannot retrieve the API key again. If you lose or forget your API key, you cannot retrieve it. Instead, generate a new API key and make the old key inactive.

Generating a long-term API key (AWS CLI)

To generate a long-term API key using the AWS CLI, use the following steps:

1. Create an IAM user that will be used with the service using the create-user command:

```
aws iam create-user \
    --user-name APIKeyUser_1
```

2. Attach the AWS managed policy to the IAM user using the attach-user-policy command.

   For Amazon Bedrock:

```
aws iam attach-user-policy --user-name APIKeyUser_1 \
    --policy-arn arn:aws:iam::aws:policy/AmazonBedrockLimitedAccess
```

   For Claude Platform on AWS:

```
aws iam attach-user-policy --user-name APIKeyUser_1 \
    --policy-arn arn:aws:iam::aws:policy/AnthropicInferenceAccess
```

   For Amazon CloudWatch:

```
aws iam attach-user-policy --user-name APIKeyUser_1 \
```

```
    --policy-arn arn:aws:iam::aws:policy/CloudWatchAPIKeyAccess
```

For Amazon CloudWatch Logs:

```
aws iam attach-user-policy --user-name APIKeyUser_1 \
    --policy-arn arn:aws:iam::aws:policy/CloudWatchLogsAPIKeyAccess
```

3. Generate the long-term API key using the create-service-specific-credential command.

For Amazon Bedrock:

```
aws iam create-service-specific-credential \
    --user-name APIKeyUser_1 \
    --service-name bedrock.amazonaws.com \
    --credential-age-days 30
```

For Claude Platform on AWS:

```
aws iam create-service-specific-credential \
    --user-name APIKeyUser_1 \
    --service-name aws-external-anthropic.amazonaws.com \
    --credential-age-days 30
```

For Amazon CloudWatch:

```
aws iam create-service-specific-credential \
    --user-name APIKeyUser_1 \
    --service-name cloudwatch.amazonaws.com \
    --credential-age-days 30
```

For Amazon CloudWatch Logs:

```
aws iam create-service-specific-credential \
    --user-name APIKeyUser_1 \
    --service-name logs.amazonaws.com \
    --credential-age-days 30
```

> **Note**
>
> The `--credential-age-days` parameter is optional. You can specify a value between 1–36600 days. If you omit this parameter, the API key does not expire.

The returned `ServiceApiKeyValue` in the response is your long-term API key for the respective service. Store the `ServiceApiKeyValue` value securely, as you cannot retrieve it later.

List long-term API keys (AWS CLI)

To list long-term API keys metadata for a specific user, use the list-service-specific-credentials command with the `--user-name` parameter:

```
aws iam list-service-specific-credentials \
    --service-name bedrock.amazonaws.com \
    --user-name APIKeyUser_1
```

> **Note**
>
> Replace `bedrock.amazonaws.com` with the appropriate service name (for example, `logs.amazonaws.com` for Amazon CloudWatch Logs or `aws-external-anthropic.amazonaws.com` for Claude Platform on AWS).

To list all long-term API keys metadata in the account, use the list-service-specific-credentials command with the `--all-users` parameter:

```
aws iam list-service-specific-credentials \
    --service-name bedrock.amazonaws.com \
    --all-users
```

Update long-term API key status (AWS CLI)

To update the status of a long-term API key, use the update-service-specific-credential command:

```
aws iam update-service-specific-credential \
    --user-name "APIKeyUser_1" \
    --service-specific-credential-id "ACCA1234EXAMPLE1234" \
```

```
    --status Inactive|Active
```

Generating a long-term API key (AWS API)

You can use the following IAM API operations to manage long-term API keys for any supported service:

- CreateServiceSpecificCredential
- ListServiceSpecificCredentials
- UpdateServiceSpecificCredential
- DeleteServiceSpecificCredential
- ResetServiceSpecificCredential

Short-term API keys (select services)

Short-term API keys are currently supported by select services.

For information on generating and using short-term API keys with Amazon Bedrock, see Generate an API key in the *Amazon Bedrock User Guide*.

For information on generating and using short-term API keys for Claude Platform on AWS, see Authentication in the *Claude Platform on AWS User Guide*.

Service-specific information

- For more information about using API keys with Amazon Bedrock, see Use an Amazon Bedrock API key in the *Amazon Bedrock User Guide*.
- For more information about using API keys with Claude Platform on AWS, see Authentication in the *Claude Platform on AWS User Guide*.
- For more information about using API keys with Amazon CloudWatch, see Setting up bearer token authentication for Metrics in the *Amazon CloudWatch User Guide*.
- For more information about using API keys with Amazon CloudWatch Logs, see Setting up bearer token authentication in the *Amazon CloudWatch Logs User Guide*.

Use IAM with Amazon Keyspaces (for Apache Cassandra)

Amazon Keyspaces (for Apache Cassandra) is a scalable, highly available, and managed Apache Cassandra-compatible database service. You can access Amazon Keyspaces through the AWS

Management Console, or programmatically. To access Amazon Keyspaces programmatically with service-specific credentials, you can use `cqlsh` or open-source Cassandra drivers. *Service-specific credentials* include a user name and password like those that Cassandra uses for authentication and access management. You can have a maximum of two sets of service-specific credentials for each supported service per user.

To access Amazon Keyspaces programmatically with AWS access keys, you can use the AWS SDK, the AWS Command Line Interface (AWS CLI) or open-source Cassandra drivers with the SigV4 plugin. To learn more, see Create and configure AWS credentials for Amazon Keyspaces in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

> **Note**
>
> If you plan to interact with Amazon Keyspaces only through the console, you don't need to generate service-specific credentials. For more information, see Accessing Amazon Keyspaces using the console in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

For more information about the permissions required to access Amazon Keyspaces, see Amazon Keyspaces (for Apache Cassandra) Identity-Based Policy Examples in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

Generating Amazon Keyspaces credentials (console)

You can use the AWS Management Console to generate Amazon Keyspaces (for Apache Cassandra) credentials for your IAM users.

To generate Amazon Keyspaces service-specific credentials (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and then choose the name of the user that requires the credentials.
3. On the **Security Credentials** tab beneath **Credentials for Amazon Keyspaces (for Apache Cassandra)**, choose **Generate credentials**.
4. Your service-specific credentials are now available. This is the only time that the password can be viewed or downloaded. You cannot recover it later. However, you can reset your password at any time. Save the user and password in a secure location, because you'll need them later.

Generating Amazon Keyspaces credentials (AWS CLI)

You can use the AWS CLI to generate Amazon Keyspaces (for Apache Cassandra) credentials for your IAM users.

To generate Amazon Keyspaces service-specific credentials (AWS CLI)

- Use the following command:
  - aws iam create-service-specific-credential

Generating Amazon Keyspaces credentials (AWS API)

You can use the AWS API to generate Amazon Keyspaces (for Apache Cassandra) credentials for your IAM users.

To generate Amazon Keyspaces service-specific credentials (AWS API)

- Complete the following operation:
  - CreateServiceSpecificCredential

Find unused AWS credentials

To increase the security of your AWS account, remove IAM user credentials (that is, passwords and access keys) that are not needed. For example, when users leave your organization or no longer need AWS access, find the credentials that they were using and ensure that they are no longer operational. Ideally, you delete credentials if they are no longer needed. You can always recreate them at a later date if the need arises. At the very least, you should change the password or deactivate the access keys so that the former users no longer have access.

Of course, the definition of *unused* can vary and usually means a credential that has not been used within a specified period of time.

Finding unused passwords

You can use the AWS Management Console to view password usage information for your users. If you have a large number of users, you can use the console to download a credential report with information about when each user last used their console password. You can also access the information from the AWS CLI or the IAM API.

To find unused passwords (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. If necessary, add the **Console last sign-in** column to the users table:
   a. Above the table on the far right, choose the settings icon ().
   b. In **Select visible columns**, select **Console last sign-in**.
   c. Choose **Confirm** to return to the list of users.
4. The **Console last sign-in** column shows the date when the user last signed in to AWS through the console. You can use this information to find users with passwords who have not signed in for more than a specified period of time. The column displays **Never** for users with passwords that have never signed in. **None** indicates users with no passwords. Passwords that have not been used recently might be good candidates for removal.

> **Important**
>
> Due to a service issue, password last used data does not include password use from May 3rd 2018 22:50 PDT to May 23rd 2018 14:08 PDT. This affects last sign-in dates shown in the IAM console and password last used dates in the IAM credential report, and returned by the GetUser API operation. If users signed in during the affected time, the password last used date that is returned is the date the user last signed in before May 3rd 2018. For users that signed in after May 23rd 2018 14:08 PDT, the returned password last used date is accurate.
> If you use password last used information to identify unused credentials for deletion, such as deleting users who did not sign in to AWS in the last 90 days, we recommend that you adjust your evaluation window to include dates after May 23rd 2018. Alternatively, if your users use access keys to access AWS programmatically you can refer to access key last used information because it is accurate for all dates.

To find unused passwords by downloading the credentials report (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Credential report**.
3. Choose **Download Report** to download a comma-separated value (CSV) file named `status_reports_<date>T<time>.csv`. The fifth column contains the `password_last_used` column with the dates or one of the following:
   - **N/A** – Users that do not have a password assigned at all.
   - **no_information** – Users that have not used their password since IAM began tracking password age on October 20, 2014.

To find unused passwords (AWS CLI)

Run the following command to find unused passwords:

- `aws iam list-users` returns a list of users, each with a `PasswordLastUsed` value. If the value is missing, then the user either has no password or the password has not been used since IAM began tracking password age on October 20, 2014.

To find unused passwords (AWS API)

Call the following operation to find unused passwords:

- `ListUsers` returns a collection of users, each of which has a <PasswordLastUsed> value. If the value is missing, then the user either has no password or the password has not been used since IAM began tracking password age on October 20, 2014.

For information about the commands to download the credentials report, see Getting credential reports (AWS CLI).

Finding unused access keys

You can use the AWS Management Console to view access key usage information for your users. If you have a large number of users, you can use the console to download a credentials report to find when each user last used their access keys. You can also access the information from the AWS CLI or the IAM API.

To find unused access keys (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. If necessary, add the **Access key last used** column to the users table:
   a. Above the table on the far right, choose the settings icon ( ).
   b. In **Select visible columns**, select **Access key last used**.
   c. Choose **Confirm** to return to the list of users.
4. The **Access key last used** column shows the number of days since the user last accessed AWS programmatically. You can use this information to find users with access keys that have not been used for more than a specified period of time. The column displays **–** for users with no access keys. Access keys that have not been used recently might be good candidates for removal.

To find unused access keys by downloading the credentials report (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Credential Report**.
3. Choose **Download Report** to download a comma-separated value (CSV) file named `status_reports_<date>T<time>.csv`. Columns 11 through 13 contain the last used date, Region, and service information for access key 1. Columns 16 through 18 contain the same information for access key 2. The value is **N/A** if the user does not have an access key or the user has not used the access key since IAM began tracking access key age on April 22, 2015.

To find unused access keys (AWS CLI)

Run the following commands to find unused access keys:

- `aws iam list-access-keys` returns information about the access keys for a user, including the `AccessKeyID`.
- `aws iam get-access-key-last-used` takes an access key ID and returns output that includes the `LastUsedDate`, the `Region` in which the access key was last used, and the

`ServiceName` of the last service requested. If `LastUsedDate` is missing, then the access key has not been used since IAM began tracking access key age on April 22, 2015.

To find unused access keys (AWS API)

Call the following operations to find unused access keys:

- `ListAccessKeys` returns a list of `AccessKeyID` values for access keys that are associated with the specified user.
- `GetAccessKeyLastUsed` takes an access key ID and returns a collection of values. Included are the `LastUsedDate`, the `Region` in which the access key was last used, and the `ServiceName` of the last service requested. If the value is missing, then either the user has no access key or the access key has not been used since IAM began tracking access key age on April 22, 2015.

For information about the commands to download the credentials report, see Getting credential reports (AWS CLI).

Generate credential reports for your AWS account

You can generate and download a *credential report* that lists all users in your account and the status of their various credentials, including passwords, access keys, and MFA devices. You can get a credential report from the AWS Management Console, the AWS SDKs and Command Line Tools, or the IAM API.

> **Note**
>
> The IAM credential report only includes the following IAM-managed credentials: passwords, the first two access keys per user, MFA devices, and X.509 signing certificates. The report does not include service-specific credentials (such as CodeCommit passwords, Amazon Bedrock long-term API keys, or Amazon CloudWatch Logs long-term API keys) or any other user access keys beyond the first two. For complete credential visibility, use the ListServiceSpecificCredentials and ListAccessKeys APIs.

You can use credential reports to assist in your auditing and compliance efforts. You can use the report to audit the effects of credential lifecycle requirements, such as password and access key updates. You can provide the report to an external auditor, or grant permissions to an auditor so that he or she can download the report directly.

You can generate a credential report as often as once every four hours. When you request a report, IAM first checks whether a report for the AWS account has been generated within the past four hours. If so, the most recent report is downloaded. If the most recent report for the account is older than four hours, or if there are no previous reports for the account, IAM generates and downloads a new report.

Topics

- Required permissions
- Understanding the report format
- Getting credential reports (console)
- Getting credential reports (AWS CLI)
- Getting credential reports (AWS API)

Required permissions

The following permissions are needed to create and download reports:

- To create a credential report: `iam:GenerateCredentialReport`
- To download the report: `iam:GetCredentialReport`

Understanding the report format

Credential reports are formatted as comma-separated values (CSV) files. You can open CSV files with common spreadsheet software to perform analysis, or you can build an application that consumes the CSV files programmatically and performs custom analysis.

The CSV file contains the following columns:

user

The friendly name of the user.

arn

The Amazon Resource Name (ARN) of the user. For more information about ARNs, see IAM ARNs.

user_creation_time

The date and time when the user was created, in ISO 8601 date-time format.

password_enabled

When the user has a password, this value is TRUE. Otherwise it is FALSE. This value is FALSE for new member accounts created as part of your organization as they have no root user credentials by default.

password_last_used

The date and time when the AWS account root user or user's password was last used to sign in to an AWS website, in ISO 8601 date-time format. AWS websites that capture a user's last sign-in time are the AWS Management Console, the AWS Discussion Forums, and the AWS Marketplace. When a password is used more than once in a 5-minute span, only the first use is recorded in this field.

- The value in this field is `no_information` in these cases:
  - The user's password has never been used.
  - There is no sign-in data associated with the password, such as when user's password has not been used after IAM started tracking this information on October 20, 2014.
- The value in this field is `N/A` (not applicable) when the user does not have a password.

Important

Due to a service issue, password last used data does not include password use from May 3rd 2018 22:50 PDT to May 23rd 2018 14:08 PDT. This affects last sign-in dates shown in the IAM console and password last used dates in the IAM credential report, and returned by the GetUser API operation. If users signed in during the affected time, the password last used date that is returned is the date the user last signed in before May 3rd 2018. For users that signed in after May 23rd 2018 14:08 PDT, the returned password last used date is accurate. If you use password last used information to identify unused credentials for deletion, such as deleting users who did not sign in to AWS in the last 90 days, we recommend that you adjust your evaluation window to include dates after May 23rd 2018. Alternatively, if your users use access keys to access AWS programmatically you can refer to access key last used information because it is accurate for all dates.

password_last_changed

The date and time when the user's password was last set, in ISO 8601 date-time format. If the user does not have a password, the value in this field is `N/A` (not applicable).

password_next_rotation

When the account has a password policy that requires password rotation, this field contains the date and time, in ISO 8601 date-time format, when the user is required to set a new password. The value for the AWS account (root) is always `not_supported`.

mfa_active

When a multi-factor authentication (MFA) device has been enabled for the user, this value is `TRUE`. Otherwise it is `FALSE`.

access_key_1_active

When the user has an access key and the access key's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`. Applies to both account root user and IAM users.

access_key_1_last_rotated

The date and time, in ISO 8601 date-time format, when the user's access key was created or last changed. If the user does not have an active access key, the value in this field is `N/A` (not applicable). Applies to both account root user and IAM users.

access_key_1_last_used_date

The date and time, in ISO 8601 date-time format, when the user's access key was most recently used to sign an AWS API request. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field. Applies to both account root user and IAM users.

The value in this field is `N/A` (not applicable) in these cases:

- The user does not have an access key.
- The access key has never been used.
- The access key has not been used after IAM started tracking this information on April 22, 2015.

access_key_1_last_used_region

The AWS Region in which the access key was most recently used. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field. Applies to both account root user and IAM users.

The value in this field is `N/A` (not applicable) in these cases:

- The user does not have an access key.
- The access key has never been used.
- The access key was last used before IAM started tracking this information on April 22, 2015.
- The last used service is not Region-specific, such as Amazon S3.

access_key_1_last_used_service

The AWS service that was most recently accessed with the access key. The value in this field uses the service's namespace—for example, `s3` for Amazon S3 and `ec2` for Amazon EC2. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field. Applies to both account root user and IAM users.

The value in this field is `N/A` (not applicable) in these cases:

- The user does not have an access key.
- The access key has never been used.
- The access key was last used before IAM started tracking this information on April 22, 2015.

access_key_2_active

When the user has a second access key and the second key's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`. Applies to both account root user and IAM users.

> **Note**
>
> Users can have up to two access keys, to make rotation easier by updating the key first and then deleting the previous key. For more information about updating access keys, see Update access keys.

access_key_2_last_rotated

The date and time, in ISO 8601 date-time format, when the user's second access key was created or last updated. If the user does not have a second active access key, the value in this field is `N/A` (not applicable). Applies to both account root user and IAM users.

access_key_2_last_used_date

The date and time, in ISO 8601 date-time format, when the user's second access key was most recently used to sign an AWS API request. When an access key is used more than once in a 15-

minute span, only the first use is recorded in this field. Applies to both account root user and IAM users.

The value in this field is `N/A` (not applicable) in these cases:

- The user does not have a second access key.
- The user's second access key has never been used.
- The user's second access key was last used before IAM started tracking this information on April 22, 2015.

access_key_2_last_used_region

The AWS Region in which the user's second access key was most recently used. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field. Applies to both account root user and IAM users. The value in this field is `N/A` (not applicable) in these cases:

- The user does not have a second access key.
- The user's second access key has never been used.
- The user's second access key was last used before IAM started tracking this information on April 22, 2015.
- The last used service is not Region-specific, such as Amazon S3.

access_key_2_last_used_service

The AWS service that was most recently accessed with the user's second access key. The value in this field uses the service's namespace—for example, `s3` for Amazon S3 and `ec2` for Amazon EC2. When an access key is used more than once in a 15-minute span, only the first use is recorded in this field. Applies to both account root user and IAM users. The value in this field is `N/A` (not applicable) in these cases:

- The user does not have a second access key.
- The user's second access key has never been used.
- The user's second access key was last used before IAM started tracking this information on April 22, 2015.

cert_1_active

When the user has an X.509 signing certificate and that certificate's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`.

cert_1_last_rotated

The date and time, in ISO 8601 date-time format, when the user's signing certificate was created or last changed. If the user does not have an active signing certificate, the value in this field is `N/A` (not applicable).

cert_2_active

When the user has a second X.509 signing certificate and that certificate's status is `Active`, this value is `TRUE`. Otherwise it is `FALSE`.

> **Note**
>
> Users can have up to two X.509 signing certificates, to make certificate rotation easier.

cert_2_last_rotated

The date and time, in ISO 8601 date-time format, when the user's second signing certificate was created or last changed. If the user does not have a second active signing certificate, the value in this field is `N/A` (not applicable).

additional_credentials_info

When the user has more than two access keys or certificates, this value is the number of additional access keys or certificates and the actions you can use to list the access keys or certificates associated with the user.

Getting credential reports (console)

You can use the AWS Management Console to download a credential report as a comma-separated values (CSV) file.

To download a credential report (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Credential report**.
3. Choose **Download Report**.

Getting credential reports (AWS CLI)

To download a credentials report (AWS CLI)

1. Generate a credentials report. AWS stores a single report. If a report exists, generating a credentials report overwrites the previous report. `aws iam generate-credential-report`
2. View the last report that was generated: `aws iam get-credential-report`

Getting credential reports (AWS API)

To download a credentials report (AWS API)

1. Generate a credentials report. AWS stores a single report. If a report exists, generating a credentials report overwrites the previous report. `GenerateCredentialReport`
2. View the last report that was generated: `GetCredentialReport`

IAM credentials for CodeCommit: Git credentials, SSH keys, and AWS access keys

CodeCommit is a managed version control service that hosts private Git repositories in the AWS cloud. To use CodeCommit, you configure your Git client to communicate with CodeCommit repositories. As part of this configuration, you provide IAM credentials that CodeCommit can use to authenticate you. IAM supports CodeCommit with three types of credentials:

- Git credentials, an IAM-generated user name and password pair you can use to communicate with CodeCommit repositories over HTTPS.
- SSH keys, a locally generated public-private key pair that you can associate with your IAM user to communicate with CodeCommit repositories over SSH.
- AWS access keys, which you can use with the credential helper included with the AWS CLI to communicate with CodeCommit repositories over HTTPS.

> **Note**
>
> You cannot use SSH keys or Git credentials to access repositories in another AWS account. To learn how to configure access to CodeCommit repositories for IAM users and groups

in another AWS account, see Configure cross-account access to an AWS CodeCommit repository using roles in the *AWS CodeCommit User Guide*.

See the following sections for more information about each option.

Use Git credentials and HTTPS with CodeCommit (recommended)

With Git credentials, you generate a static user name and password pair for your IAM user, and then use those credentials for HTTPS connections. You can also use these credentials with any third-party tool or integrated development environment (IDE) that supports static Git credentials.

Because these credentials are universal for all supported operating systems and compatible with most credential management systems, development environments, and other software development tools, this is the recommended method. You can reset the password for Git credentials at any time. You can also make the credentials inactive or delete them if you no longer need them.

Note

You cannot choose your own user name or password for Git credentials. IAM generates these credentials for you to help ensure they meet the security standards for AWS and secure repositories in CodeCommit. You can download the credentials only once, at the time they are generated. Make sure that you save the credentials in a secure location. If necessary, you can reset the password at any time, but doing so invalidates any connections configured with the old password. You must reconfigure connections to use the new password before you can connect.

See the following topics for more information:

- To create an IAM user, see Create an IAM user in your AWS account.
- To generate and use Git credentials with CodeCommit, see For HTTPS Users Using Git Credentials in the *AWS CodeCommit User Guide*.

> **Note**
>
> Changing the name of an IAM user after generating Git credentials does not change the user name of the Git credentials. The user name and password remain the same and are still valid.

To update service specific credentials

1. Create a second service-specific credential set in addition to the set currently in use.
2. Update all of your applications to use the new set of credentials and validate that the applications are working.
3. Change the state of the original credentials to "Inactive".
4. Ensure that all of your applications are still working.
5. Delete the inactive service-specific credentials.

Use SSH keys and SSH with CodeCommit

With SSH connections, you create public and private key files on your local machine that Git and CodeCommit use for SSH authentication. You associate the public key with your IAM user and store the private key on your local machine. See the following topics for more information:

- To create an IAM user, see Create an IAM user in your AWS account.
- To create an SSH public key and associate it with an IAM user, see For SSH Connections on Linux, macOS, or Unix or see For SSH Connections on Windows in the *AWS CodeCommit User Guide*.

> **Note**
>
> The public key must be encoded in ssh-rsa format or PEM format. The minimum bit-length of the public key is 2048 bits, and the maximum length is 16384 bits. This is separate from the size of the file you upload. For example, you can generate a 2048-bit key, and the resulting PEM file is 1679 bytes long. If you provide your public key in another format or size, you will see an error message stating that the key format is not valid.

Use HTTPS with the AWS CLI credential helper and CodeCommit

As an alternative to HTTPS connections with Git credentials, you can allow Git to use a cryptographically signed version of your IAM user credentials or Amazon EC2 instance role whenever Git needs to authenticate with AWS to interact with CodeCommit repositories. This is the only connection method for CodeCommit repositories that does not require an IAM user. This is also the only method that works with federated access and temporary credentials. See the following topics for more information:

- To learn more about federated access, see Identity providers and federation into AWS and Access to externally authenticated users (identity federation).
- To learn more about temporary credentials, see Temporary security credentials in IAM and Temporary Access to CodeCommit Repositories.

The AWS CLI credential helper is not compatible with other credential helper systems, such as Keychain Access or Windows Credential Management. There are additional configuration considerations when you configure HTTPS connections with the credential helper. For more information, see For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper or HTTPS Connections on Windows with the AWS CLI Credential Helper in the *AWS CodeCommit User Guide*.

Manage server certificates in IAM

To enable HTTPS connections to your website or application in AWS, you need an SSL/TLS *server certificate*. For certificates in a Region supported by AWS Certificate Manager (ACM), we recommend that you use ACM to provision, manage, and deploy your server certificates. In unsupported Regions, you must use IAM as a certificate manager. To learn which Regions ACM supports, see AWS Certificate Manager endpoints and quotas in the *AWS General Reference*.

> **Important**
>
> ACM is the preferred tool to provision, manage, and deploy your server certificates. With ACM you can request a certificate or deploy an existing ACM or external certificate to AWS resources. Certificates provided by ACM are free and automatically renew. In a supported Region, you can use ACM to manage server certificates from the console or programmatically. For more information about using ACM, see the *AWS Certificate Manager User Guide*. For more information about requesting an ACM certificate, see Request a Public Certificate or Request a Private Certificate in the *AWS Certificate Manager User Guide*.

For more information about importing third-party certificates into ACM, see Importing Certificates in the *AWS Certificate Manager User Guide*.

Use IAM as a certificate manager only when you must support HTTPS connections in a Region that is not supported by ACM. IAM securely encrypts your private keys and stores the encrypted version in IAM SSL certificate storage. IAM supports deploying server certificates in all Regions, but you must obtain your certificate from an external provider for use with AWS. You cannot upload an ACM certificate to IAM. Additionally, you cannot manage your certificates from the IAM Console.

For more information about uploading third-party certificates to IAM, see the following topics.

Topics

- Upload a server certificate (AWS API)
- AWS API operations for server certificates
- Troubleshoot server certificates

Upload a server certificate (AWS API)

To upload a server certificate to IAM, you must provide the certificate and its matching private key. When the certificate is not self-signed, you must also provide a certificate chain. (You don't need a certificate chain when uploading a self-signed certificate.) Before you upload a certificate, ensure that you have all these items and that they meet the following criteria:

- The certificate must be valid at the time of upload. You cannot upload a certificate before its validity period begins (the certificate's `NotBefore` date) or after it expires (the certificate's `NotAfter` date).
- The private key must be unencrypted. You cannot upload a private key that is protected by a password or passphrase. For help decrypting an encrypted private key, see Troubleshoot server certificates.
- The certificate, private key, and certificate chain must all be PEM-encoded. For help converting these items to PEM format, see Troubleshoot server certificates.

To use the IAM API to upload a certificate, send an UploadServerCertificate request. The following example shows how to do this with the AWS Command Line Interface (AWS CLI). The example assumes the following:

- The PEM-encoded certificate is stored in a file named `Certificate.pem`.
- The PEM-encoded certificate chain is stored in a file named `CertificateChain.pem`.
- The PEM-encoded, unencrypted private key is stored in a file named `PrivateKey.pem`.
- (Optional) You want to tag the server certificate with a key–value pair. For example, you might add the tag key `Department` and the tag value `Engineering` to help you identify and organize your certificates.

To use the following example command, replace these file names with your own. Replace *`ExampleCertificate`* with a name for your uploaded certificate. If you want to tag the certificate, replace the *`ExampleKey`* and *`ExampleValue`* tag key-value pair with your own values. Type the command on one continuous line. The following example includes line breaks and extra spaces to make it easier to read.

```
aws iam upload-server-certificate --server-certificate-name ExampleCertificate
                                  --certificate-body file://Certificate.pem
                                  --certificate-chain file://CertificateChain.pem
                                  --private-key file://PrivateKey.pem
                                  --tags '{"Key": "ExampleKey", "Value":
 "ExampleValue"}'
```

When the preceding command is successful, it returns metadata about the uploaded certificate, including its Amazon Resource Name (ARN), its friendly name, its identifier (ID), its expiration date, tags, and more.

> **Note**
>
> If you are uploading a server certificate to use with Amazon CloudFront, you must specify a path using the `--path` option. The path must begin with `/cloudfront` and must include a trailing slash (for example, `/cloudfront/test/`).

To use the AWS Tools for Windows PowerShell to upload a certificate, use Publish-IAMServerCertificate.

AWS API operations for server certificates

Use the following commands to view, tag, rename, and delete server certificates.

- Use GetServerCertificate to retrieve a certificate. This request returns the certificate, the certificate chain (if one was uploaded), and metadata about the certificate.

  > **Note**
  >
  > You cannot download or retrieve a private key from IAM after you upload it.

- Use Get-IAMServerCertificate to retrieve a certificate.
- Use ListServerCertificates to list your uploaded server certificates. The request returns a list that contains metadata about each certificate.
- Use Get-IAMServerCertificates to list your uploaded server certificates.
- Use TagServerCertificate to tag an existing server certificate.
- Use UntagServerCertificate to untag a server certificate.
- Use UpdateServerCertificate to rename a server certificate or update its path.

  The following example shows how to do this with the AWS CLI.

  To use the following example command, replace the old and new certificate names and the certificate path, and type the command on one continuous line. The following example includes line breaks and extra spaces to make it easier to read.

  ```
  aws iam update-server-certificate --server-certificate-name ExampleCertificate
                                    --new-server-certificate-
  name CloudFrontCertificate
                                    --new-path /cloudfront/
  ```

  To use the AWS Tools for Windows PowerShell to rename a server certificate or update its path, use Update-IAMServerCertificate.

- Use DeleteServerCertificate to delete a server certificate.

  To use the AWS Tools for Windows PowerShell to delete a server certificate, use Remove-IAMServerCertificate.

Troubleshoot server certificates

Before you can upload a certificate to IAM, you must make sure that the certificate, private key, and certificate chain are all PEM-encoded. You must also ensure that the private key is unencrypted. See the following examples.

Example PEM-encoded certificate

```
-----BEGIN CERTIFICATE-----
Base64-encoded certificate
-----END CERTIFICATE-----
```

Example PEM-encoded, unencrypted private key

```
-----BEGIN RSA PRIVATE KEY-----
Base64-encoded private key
-----END RSA PRIVATE KEY-----
```

Example PEM-encoded certificate chain

A certificate chain contains one or more certificates. You can use a text editor, the copy command in Windows, or the Linux cat command to concatenate your certificate files into a chain. When you include multiple certificates, each certificate must certify the preceding certificate. You accomplish this by concatenating the certificates, including the root CA certificate last.

The following example contains three certificates, but your certificate chain might contain more or fewer certificates.

```
-----BEGIN CERTIFICATE-----
Base64-encoded certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Base64-encoded certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Base64-encoded certificate
-----END CERTIFICATE-----
```

If these items are not in the right format for uploading to IAM, you can use OpenSSL to convert them to the right format.

To convert a certificate or certificate chain from DER to PEM

Use the OpenSSL **x509** command, as in the following example. In the following example command, replace *Certificate.der* with the name of the file that contains your DER-encoded certificate. Replace *Certificate.pem* with the preferred name of the output file to contain the PEM-encoded certificate.

```
openssl x509 -inform DER -in Certificate.der -outform PEM -out Certificate.pem
```

To convert a private key from DER to PEM

Use the OpenSSL **rsa** command, as in the following example. In the following example command, replace *PrivateKey.der* with the name of the file that contains your DER-encoded private key. Replace *PrivateKey.pem* with the preferred name of the output file to contain the PEM-encoded private key.

```
openssl rsa -inform DER -in PrivateKey.der -outform PEM -out PrivateKey.pem
```

To decrypt an encrypted private key (remove the password or passphrase)

Use the OpenSSL **rsa** command, as in the following example. To use the following example command, replace *EncryptedPrivateKey.pem* with the name of the file that contains your encrypted private key. Replace *PrivateKey.pem* with the preferred name of the output file to contain the PEM-encoded unencrypted private key.

```
openssl rsa -in EncryptedPrivateKey.pem -out PrivateKey.pem
```

To convert a certificate bundle from PKCS#12 (PFX) to PEM

Use the OpenSSL **pkcs12** command, as in the following example. In the following example command, replace *CertificateBundle.p12* with the name of the file that contains your PKCS#12-encoded certificate bundle. Replace *CertificateBundle.pem* with the preferred name of the output file to contain the PEM-encoded certificate bundle.

```
openssl pkcs12 -in CertificateBundle.p12 -out CertificateBundle.pem -nodes
```

To convert a certificate bundle from PKCS#7 to PEM

Use the OpenSSL **pkcs7** command, as in the following example. In the following example command, replace *CertificateBundle.p7b* with the name of the file that contains your PKCS#7-encoded certificate bundle. Replace *CertificateBundle.pem* with the preferred name of the output file to contain the PEM-encoded certificate bundle.

```
openssl pkcs7 -in CertificateBundle.p7b -print_certs -out CertificateBundle.pem
```

IAM user groups

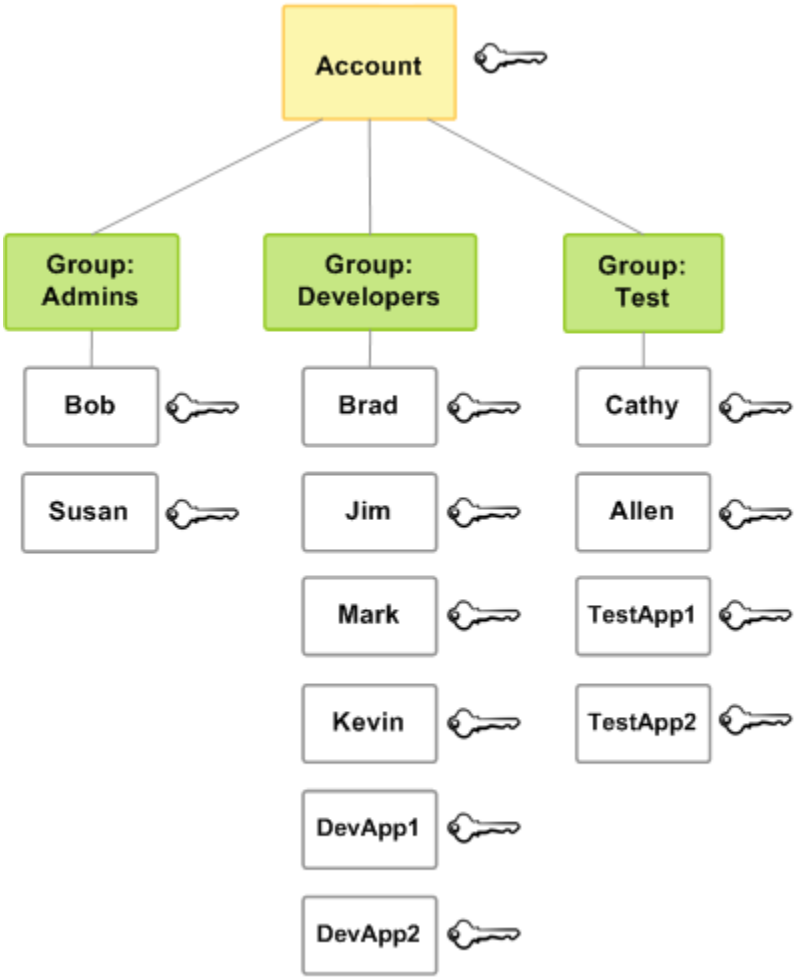
An IAM *user group* is a collection of IAM users. User groups let you specify permissions for multiple users, which can make it easier to manage the permissions for those users. For example, you could have a user group called *Admins* and give that user group typical administrator permissions. Any user in that user group automatically has *Admins* group permissions. If a new user joins your organization and needs administrator privileges you can assign the appropriate permissions by adding the user to the *Admins* user group. If a person changes jobs in your organization, instead of editing that user's permissions you can remove them from the old IAM groups and add them to the appropriate new IAM groups.

You can attach an identity-based policy to a user group so that all of the users in the user group receive the policy's permissions. You cannot identify a user group as a `Principal` in a policy (such as a resource-based policy) because groups relate to permissions, not authentication, and principals are authenticated IAM entities. For more information about policy types, see Identity-based policies and resource-based policies.

Here are some important characteristics of IAM groups:

- A user group can contain many users, and a user can belong to multiple user groups.
- User groups can't be nested; they can contain only users, not other IAM groups.
- There is no default user group that automatically includes all users in the AWS account. If you want to have a user group like that, you must create it and assign each new user to it.
- The number and size of IAM resources in an AWS account, such as the number of groups, and the number of groups that a user can be a member of, are limited. For more information, see IAM and AWS STS quotas.

The following diagram shows a simple example of a small company. The company owner creates an `Admins` user group for users to create and manage other users as the company grows. The `Admins` user group creates a `Developers` user group and a `Test` user group. Each of these IAM groups consists of users (humans and applications) that interact with AWS (Jim, Brad, DevApp1, and so on). Each user has an individual set of security credentials. In this example, each user belongs to a single user group. However, users can belong to multiple IAM groups.



After you create a user group, you can View IAM groups, Attach a policy to an IAM user group, and Rename an IAM user group.

Create IAM groups

> **Note**
>
> As a best practice, we recommend that you require human users to use federation with an identity provider to access AWS using temporary credentials. If you follow the best practices, you are not managing IAM users and groups. Instead, your users and groups are managed outside of AWS and are able to access AWS resources as a *federated identity*. A federated identity is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. Federated identities use the groups defined by their identity provider. If you are using AWS IAM Identity Center, see Manage identities in IAM Identity Center in the *AWS IAM Identity Center User Guide* for information about creating users and groups in IAM Identity Center.

You create IAM groups to manage access permissions for multiple users with similar roles or responsibilities. By attaching policies to these groups, you can grant or revoke permissions for entire sets of users. This simplifies your maintenance of security policies, as changes you make to a group's permissions are automatically applied to all members of that group, ensuring consistent access control. After you create the group, give the group permissions based on the type of work that you expect the IAM users in the group to do, then add the IAM users to the group.

For information about the permissions required to create an IAM group, see Permissions required to access IAM resources.

To create an IAM group and attach policies

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **User groups** and then choose **Create group**.
3. For **User group name**, type the name of the group.

> **Note**
>
> The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas. Group names can be a combination of up to 128 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), underscore (_), and hyphen (-). Names must be unique within an account. They aren't distinguished by case. For example, you cannot create groups named both **`ADMINS`** and **`admins`**.

4. In the list of users, select the check box for each user that you want to add to the group.
5. In the list of policies, select the check box for each policy that you want to apply to all members of the group.
6. Choose **Create group**.

AWS CLI

Run the following command:

- aws iam create-group

API

Call the following operation:

- CreateGroup

View IAM groups

You can list all the IAM groups in your account, list the users in a user group, and list the IAM groups a user belongs to. If you use the CLI or API, you can list all the IAM groups with a particular path prefix.

Console

To list all IAM groups in your account:

- In the navigation pane , choose **User groups**.

To list the IAM users in a specific IAM group:

- In the navigation pane, choose **User groups**. Then choose the name of the group to open the group details page. Review the **Users** tab to see the group membership.

To list all the IAM groups that a user is in:

- In the navigation pane, choose **Users**. Then choose the user name to open the user details page. Choose the **Groups** tab to see a list of the groups to which the user belongs.

AWS CLI

To list all IAM groups in your account:

- aws iam list-groups

To list the users in a specific IAM group:

- aws iam get-group

To list all the IAM groups that a user is in:

- aws iam list-groups-for-user

API

To list all IAM groups in your account:

- ListGroups

To list the users in a specific IAM group:

- GetGroup

To list all the IAM groups that a user is in:

- ListGroupsForUser

Edit users in IAM groups

Use IAM groups to apply the same permissions policies across multiple users at once. You can then add users to or remove users from an IAM group. This is useful as people enter and leave your organization.

Review policy access

Before you remove a group, use the group details page to review the members (IAM users) of the group, the policies attached to the group on the **Permissions** tab and review recent service-level activity using the **Last Accessed** tab. This helps prevent unintentionally removing access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Add an IAM user to an IAM group

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **User groups** and then choose the name of the group.
3. Choose the **Users** tab and then choose **Add users**. Select the check box next to the users you want to add.
4. Choose **Add users**.

AWS CLI

Run the following command:

- `aws iam add-user-to-group`

API

Call the following operation:

- `AddUserToGroup`

Remove an IAM user from an IAM group

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **User groups** and then choose the name of the group.
3. Choose the **Users** tab. Select the check box next to the users you want to remove and then choose **Remove users**.

AWS CLI

Run the following command:

- `aws iam remove-user-from-group`

API

Call the following operation:

- `RemoveUserFromGroup`

Attach a policy to an IAM user group

You can attach an AWS managed policy—that is, a prewritten policy provided by AWS—to a user group, as explained in the following steps. To attach a customer managed policy—that is, a policy with custom permissions that you create—you must first create the policy. For information about creating customer managed policies, see Define custom IAM permissions with customer managed policies.

For more information about permissions and policies, see Access management for AWS resources.

To attach a policy to an IAM group

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **User groups** and then choose the name of the group.
3. Choose the **Permissions** tab.
4. Choose **Add permissions** and then choose **Attach policies**.
5. The current policies attached to the user group are displayed in the **Current permissions policies** list. In the list of **Other permissions policies**, select the check box next to the names of the policies to attach. You can use the search box to filter the list of policies by type and policy name.
6. Select the policy you want to attach to your IAM group and choose **Attach policies**.

AWS CLI

Run the following command:

- `aws iam attach-group-policy`

API

Call the following operation:

- `AttachGroupPolicy`

Rename an IAM user group

When you change a user group's name or path, the following happens:

- Any policies attached to the user group stay with the group under the new name.
- The user group retains all its users under the new name.
- The unique ID for the user group remains the same. For more information about unique IDs, see Unique identifiers.

IAM does not automatically update policies that refer to the user group as a resource to use the new name. Therefore, you must be careful when you rename a user group. Before you rename your user group, you must manually check all of your policies to find any policies where that user group is mentioned by name. For example, let's say Bob is the manager of the testing part of the organization. Bob has a policy attached to his IAM user entity that lets him add and remove users from the Test user group. If an administrator changes the name of the user group (or changes the

group path), the administrator must also update the policy attached to Bob to use the new name or path. Otherwise Bob won't be able to add and remove users from the user group.

To find policies that refer to an IAM group as a resource:

1. From the navigation pane of the IAM console, choose **Policies**.
2. Sort by the **Type** column to find your **Customer managed** custom policies.
3. Choose the policy name of the policy to edit.
4. Choose the **Permissions** tab, and then choose **Summary**.
5. Choose **IAM** from the list of services, if it exists.
6. Look for the name of your user group in the **Resource** column.
7. Choose **Edit** to change the name of your user group in the policy.

To change the name of an IAM user group

Console

1. In the navigation pane, select **User groups** and then select the group name.
2. Choose **Edit**. Type the new user group name and then choose **Save changes**.

AWS CLI

Run the following command:

- aws iam update-group

API

Call the following operation:

- UpdateGroup

Delete an IAM group

When you delete an IAM group in the console, the console automatically removes all group members, detaches all attached managed policies, and deletes all inline policies. However, because

IAM doesn't automatically delete policies that refer to the IAM group as a resource, you must be careful when you delete an IAM group. Before you delete your IAM group, manually review your policies to find any policies that mention the group by name. For example, John, the Test Team manager, has a policy attached to his IAM user entity that lets him add and remove users from the Test user group. If an administrator deletes the group, the administrator must also delete the policy attached to John. Otherwise, if the administrator recreates the deleted group and gives it the same name, John's permissions remain in place, even if he left the Test Team.

In contrast, when you use the CLI, SDK, or API to delete a user group, you remove the users in the group first. Then you delete any inline policies embedded in the IAM group. Next, you detach any managed policies that are attached to the group. Then you delete the IAM group itself.

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **User groups**.
3. In the list of IAM groups, select the check box next to the names of the IAM groups to delete. You can use the search box to filter the list of IAM groups by type, permissions, and group name.
4. Choose **Delete**.
5. In the confirmation box, if you want to delete a single group, type the group name and choose **Delete**. If you want to delete multiple groups, type the number of IAM group to delete followed by **`user groups`** and choose **Delete**. For example, if you want to delete three groups, type **`3 user groups`**.

AWS CLI

1. Remove all users from the IAM group.
   - aws iam get-group (to get the list of users in the IAM group), and aws iam remove-user-from-group (to remove a user from the IAM group)
2. Delete all inline policies embedded in the IAM group.
   - aws iam list-group-policies (to get a list of the IAM group's inline policies), and aws iam delete-group-policy (to delete the IAM group's inline policies)
3. Detach all managed policies attached to the IAM group.

- aws iam list-attached-group-policies (to get a list of the managed policies attached to the IAM group), and aws iam detach-group-policy (to detach a managed policy from the IAM group)

4. Delete the IAM group.

   - aws iam delete-group

API

1. Remove all users from the IAM group.

   - GetGroup (to get the list of users in the IAM group) and RemoveUserFromGroup (to remove a user from the IAM group)

2. Delete all inline policies embedded in the IAM group.

   - ListGroupPolicies (to get a list of the IAM group's inline policies) and DeleteGroupPolicy (to delete the IAM group's inline policies)

3. Detach all managed policies attached to the IAM group.

   - ListAttachedGroupPolicies (to get a list of the managed policies attached to the IAM group) and DetachGroupPolicy (to detach a managed policy from the IAM group)

4. Delete the IAM group.

   - DeleteGroup

IAM roles

An IAM *role* is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

You can use roles to delegate access to users, applications, or services that don't normally have access to your AWS resources. For example, you might want to grant users in your AWS account

access to resources they don't usually have, or grant users in one AWS account access to resources in another account. Or you might want to allow a mobile app to use AWS resources, but not want to embed AWS keys within the app (where they can be difficult to update and where users can potentially extract them). Sometimes you want to give AWS access to users who already have identities defined outside of AWS, such as in your corporate directory. Or, you might want to grant access to your account to third parties so that they can perform an audit on your resources.

For these scenarios, you can delegate access to AWS resources using an *IAM role*. This section introduces roles and the different ways you can use them, when and how to choose among approaches, and how to create, manage, switch to (or assume), and delete roles.

Account access manager is an IAM feature that lets you centrally assign IAM roles across your organization's accounts to IAM Identity Center users and groups. Account access manager gives your workforce access through IAM roles with their full feature set. You can use it alongside IAM Identity Center permission sets or on its own.

> **Note**
>
> When you first create your AWS account, no roles are created by default. As you add services to your account, they may add service-linked roles to support their use cases.
> A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
> Before you can delete service-linked roles you must first delete their related resources. This protects your resources because you can't inadvertently remove permission to access the resources.
> For information about which services support using service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Topics

- When to create an IAM user (instead of a role)
- Roles terms and concepts
- Additional resources

- The confused deputy problem
- Common scenarios for IAM roles
- IAM role creation
- IAM role management
- Methods to assume a role

When to create an IAM user (instead of a role)

We recommend you only use IAM users for use cases not supported by identity federation. Some of the use cases include the following:

- **Workloads that cannot use IAM roles** – You might run a workload from a location that needs to access AWS. In some situations, you can't use IAM roles to provide temporary credentials, such as for WordPress plugins. In these situations, use IAM user long-term access keys for that workload to authenticate to AWS.
- **Third-party AWS clients** – If you are using tools that don't support access with IAM Identity Center, such as third-party AWS clients or vendors that aren't hosted on AWS, use IAM user long-term access keys.
- **AWS CodeCommit access** – If you are using CodeCommit to store your code, you can use an IAM user with either SSH keys or service-specific credentials for CodeCommit to authenticate to your repositories. We recommend that you do this in addition to using a user in IAM Identity Center for normal authentication. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. To give users access to your CodeCommit repositories without configuring IAM users, you can configure the **git-remote-codecommit** utility. For more information about IAM and CodeCommit, see IAM credentials for CodeCommit: Git credentials, SSH keys, and AWS access keys. For more information about configuring the **git-remote-codecommit** utility, see Connecting to AWS CodeCommit repositories with rotating credentials in the *AWS CodeCommit User Guide*.
- **Amazon Keyspaces (for Apache Cassandra) access** – In a situation where you are unable to use users in IAM Identity Center, such as for testing purposes for Cassandra compatibility, you can use an IAM user with service-specific credentials to authenticate with Amazon Keyspaces. Users in IAM Identity Center are the people in your workforce who need access to your AWS accounts or to your cloud applications. You can also connect to Amazon Keyspaces using temporary credentials. For more information, see Using temporary credentials to connect to

Amazon Keyspaces using an IAM role and the SigV4 plugin in the *Amazon Keyspaces (for Apache Cassandra) Developer Guide*.

- **Emergency access** – In a situation where you can't access your identity provider and you must take action in your AWS account. Establishing emergency access IAM users can be part of your resiliency plan. We recommend that the emergency user credentials be tightly controlled and secured using multi-factor authentication (MFA).

Roles terms and concepts

Here are some basic terms to help you get started with roles.

Role

An IAM identity that you can create in your account that has specific permissions. An IAM role has some similarities to an IAM user. Roles and users are both AWS identities with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

Roles can be assumed by the following:

- An IAM user in the same AWS account or another AWS account
- IAM roles in the same account
- Service principals, for use with AWS services and features like:
  - Services that allow you to run code on compute services, like Amazon EC2 or AWS Lambda
  - Features that perform actions to your resources on your behalf, like Amazon S3 object replication
  - Services that deliver temporary security credentials to your applications that run outside of AWS, such as IAM Roles Anywhere or Amazon ECS Anywhere
- An external user authenticated by an external identity provider (IdP) service that is compatible with SAML 2.0 or OpenID Connect

AWS service role

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

AWS service-linked role

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

> **Note**
>
> If you are already using a service when it begins supporting service-linked roles, you might receive an email announcing a new role in your account. In this case, the service automatically created the service-linked role in your account. You don't need to take any action to support this role, and you should not manually delete it. For more information, see A new role appeared in my AWS account.

For information about which services support using service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service. For more information, see Create a service-linked role.

Role chaining

Role chaining is when you use a role to assume a second role. You can perform role chaining through the AWS Management Console by switching roles, the AWS CLI, or API. For example, `RoleA` has permission to assume `RoleB`. You can enable User1 to assume `RoleA` by using their long-term user credentials in the AssumeRole API operation. This returns `RoleA` short-term credentials. With role chaining, you can use `RoleA`'s short-term credentials to enable User1 to assume `RoleB`.

When you assume a role, you can pass a session tag and set the tag as transitive. Transitive session tags are passed to all subsequent sessions in a role chain. To learn more about session tags, see Pass session tags in AWS STS.

Role chaining limits your AWS Management Console, AWS CLI or AWS API role session to a maximum of one hour. It applies regardless of the maximum session duration configured for individual roles. When you use the AssumeRole API operation to assume a role, you can specify the duration of your role session with the `DurationSeconds` parameter. You can specify a parameter value of up to 43200 seconds (12 hours), depending on the maximum session duration setting for your role. However, if you assume a role using role chaining and provide a `DurationSeconds` parameter value greater than one hour, the operation fails.

For information about switching to a role in the AWS Management Console, see Switch from a user to an IAM role (console).

Delegation

The granting of permissions to someone to allow access to resources that you control. Delegation involves setting up a trust between two accounts. The first is the account that owns the resource (the trusting account). The second is the account that contains the users that need to access the resource (the trusted account). The trusted and trusting accounts can be any of the following:

- The same account.
- Separate accounts that are both under your organization's control.
- Two accounts owned by different organizations.

To delegate permission to access a resource, you create an IAM role in the trusting account that has two policies attached. The *permissions policy* grants the user of the role the needed permissions to carry out the intended tasks on the resource. The *trust policy* specifies which trusted account members are allowed to assume the role.

When you create a trust policy, you cannot specify a wildcard (*) as part of an ARN in the principal element. The trust policy is attached to the role in the trusting account, and is one-half of the permissions. The other half is a permissions policy attached to the user in the trusted account that allows that user to switch to, or assume the role. A user who assumes a role temporarily gives up his or her own permissions and instead takes on the permissions of the role. When the user exits, or stops using the role, the original user permissions are restored. An additional parameter called external ID helps ensure secure use of roles between accounts that are not controlled by the same organization.

Trust policy

A JSON policy document in which you define the principals that you *trust* to assume the role. A role trust policy is a required resource-based policy that is attached to a role in IAM. The

principals that you can specify in the trust policy include users, roles, accounts, and services. For more information, see How to use trust policies in IAM roles in *AWS Security Blog*.

Role for cross-account access

A role that grants access to resources in one account to a trusted principal in a different account. Roles are the primary way to grant cross-account access. However, some AWS services allow you to attach a policy directly to a resource (instead of using a role as a proxy). These are called resource-based policies, and you can use them to grant principals in another AWS account access to the resource. Some of these resources include Amazon Simple Storage Service (S3) buckets, Amazon Glacier vaults, Amazon Simple Notification Service (SNS) topics, and Amazon Simple Queue Service (SQS) queues. To learn which services support resource-based policies, see AWS services that work with IAM. For more information about resource-based policies, see Cross account resource access in IAM.

Additional resources

The following resources can help you learn more about IAM terminology related to IAM roles.

- **Principals** are entities in AWS that can perform actions and access resources. A principal can be an AWS account root user, an IAM user, or a role. A principal that represents the identity of an AWS service is a service principal. Use the Principal element in role trust policies to define the principals that you trust to assume the role.

  For more information and examples of principals you can allow to assume a role, see AWS JSON policy elements: Principal.
- **Identity federation** creates a trust relationship between an external identity provider and AWS. You can use your existing OpenID Connect (OIDC) or Security Assertion Markup Language (SAML) 2.0 provider to manage who can access AWS resources. When you use OIDC and SAML 2.0 to configure a trust relationship between these external identity providers and AWS , the user is assigned to an IAM role. The user also receives temporary credentials that allow the user to access your AWS resources.

  For more information about federated principals, see Identity providers and federation into AWS.
- **Federated principals** are existing identities from Directory Service, your enterprise user directory, or an OIDC provider. AWS assigns a role to a federated principal when access is requested through an identity provider.

For more information about SAML and OIDC federated principals, see Federated user sessions and roles.

- **Permissions policies** are identity-based policies that define what actions and resources the role can use. The document is written according to the rules of the IAM policy language.

  For more information, see IAM JSON policy reference.

- **Permissions boundaries** are an advanced feature in which you use policies to limit the maximum permissions that an identity-based policy can grant to a role. You cannot apply a permissions boundary to a service-linked role.

  For more information, see Permissions boundaries for IAM entities.

The confused deputy problem

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. To prevent this, AWS provides tools that help you protect your account if you provide third parties (known as *cross-account*) or other AWS services (known as *cross-service*) access to resources in your account.

At times, you might need to give a third party access to your AWS resources (delegate access). For example, you decide to hire a third-party company called Example Corp to monitor your AWS account and help optimize costs. In order to track your daily spending, Example Corp needs to access your AWS resources. Example Corp also monitors many other AWS accounts for other customers. You can use an IAM role to establish a trusted relationship between your AWS account and the Example Corp account. One important aspect of this scenario is the *external ID*, an optional identifier that you can use in an IAM role trust policy to designate who can assume the role. The primary function of the external ID is to address and prevent the confused deputy problem.

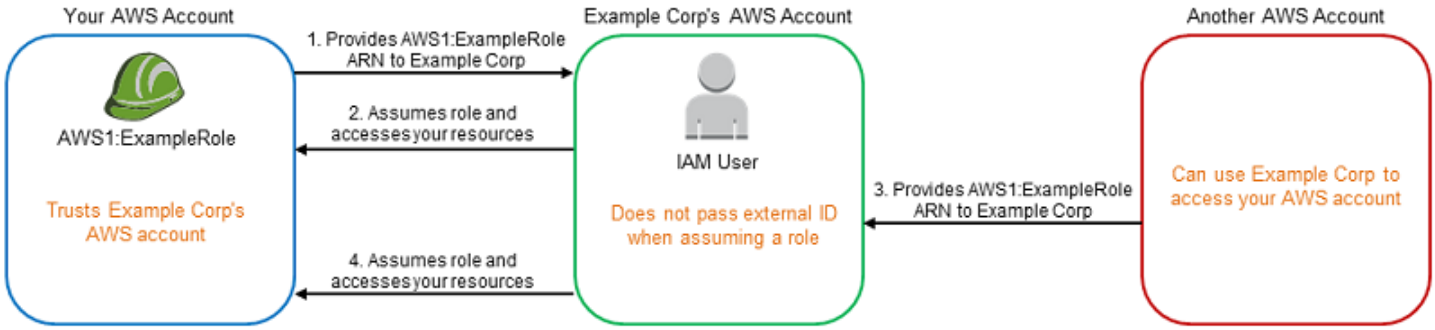
Some AWS services (calling services) use their AWS service principal to access AWS resources from other AWS services (called services). In some of these service interactions you can configure calling services to communicate with the resources from a called service in a different AWS account. An example of this is configuring AWS CloudTrail to write to a central Amazon S3 bucket which is located in a different AWS account. The calling service, CloudTrail, is granted access to your S3 bucket using the S3 bucket's policy by adding an allow statement for `cloudtrail.amazonaws.com`.

When an AWS service principal from a calling service is accessing a resource from a called service, the resource policy from the called service is only authorizing the AWS service principal, and not the actor who configured the calling service. For example, an S3 bucket that trusts the CloudTrail service principal with no conditions could receive CloudTrail logs from the AWS accounts that a trusted admin configures, but also CloudTrail logs from an unauthorized actor in their AWS account, if they know the name of the S3 bucket.

The confused deputy problem arises when an actor uses the trust of an AWS service's service principal to gain access to resources that they are not intended to have access to.

Cross-account confused deputy prevention

The following diagram illustrates the cross-account confused deputy problem.



This scenario assumes the following:

- **AWS1** is your AWS account.
- **AWS1:ExampleRole** is a role in your account. This role's trust policy trusts Example Corp by specifying Example Corp's AWS account as the one that can assume the role.

Here's what happens:

1. When you start using Example Corp's service, you provide the ARN of **AWS1:ExampleRole** to Example Corp.
2. Example Corp uses that role ARN to obtain temporary security credentials to access resources in your AWS account. In this way, you are trusting Example Corp as a "deputy" that can act on your behalf.
3. Another AWS customer also starts using Example Corp's service, and this customer also provides the ARN of **AWS1:ExampleRole** for Example Corp to use. Presumably the other customer learned or guessed the **AWS1:ExampleRole**, which isn't a secret.

4. When the other customer asks Example Corp to access AWS resources in (what it claims to be) its account, Example Corp uses **AWS1:ExampleRole** to access resources in your account.

This is how the other customer could gain unauthorized access to your resources. Because this other customer was able to trick Example Corp into unwittingly acting on your resources, Example Corp is now a "confused deputy."

Example Corp can address the confused deputy problem by requiring that you include the `ExternalId` condition check in the role's trust policy. Example Corp generates a unique `ExternalId` value for each customer and uses that value in its request to assume the role. The `ExternalId` value must be unique among Example Corp's customers and controlled by Example Corp, not its customers. This is why you get it from Example Corp and you don't come up with it on your own. This prevents Example Corp from being a confused deputy and granting access to another account's AWS resources.

In our scenario, imagine Example Corp's unique identifier for you is 12345, and its identifier for the other customer is 67890. These identifiers are simplified for this scenario. Generally, these identifiers are GUIDs. Assuming that these identifiers are unique among Example Corp's customers, they are sensible values to use for the external ID.

Example Corp gives the external ID value of 12345 to you. You must then add a `Condition` element to the role's trust policy that requires the `sts:ExternalId` value to be 12345, like this:

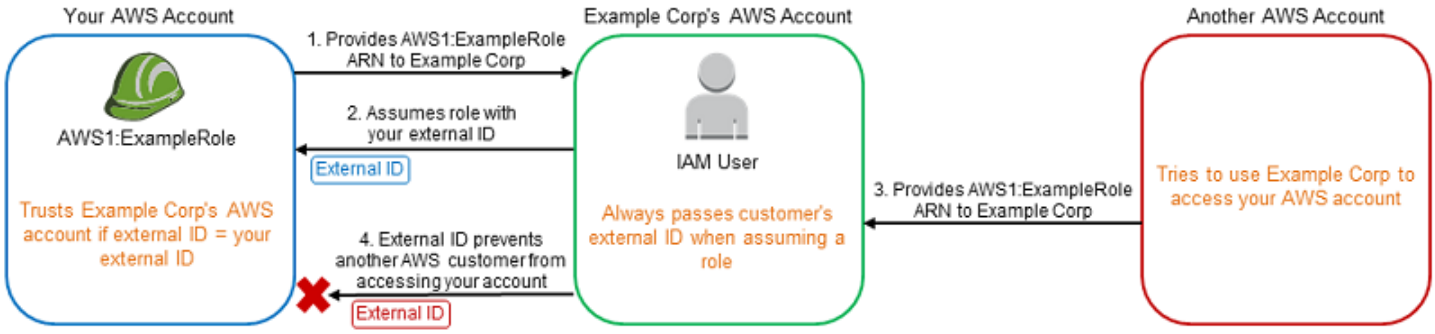
JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "AWS": "Example Corp's AWS Account ID"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "sts:ExternalId": "12345"
      }
    }
  }
```

```
}
```

The Condition element in this policy allows Example Corp to assume the role only when the AssumeRole API call includes the external ID value of 12345. Example Corp makes sure that whenever it assumes a role on behalf of a customer, it always includes that customer's external ID value in the AssumeRole call. Even if another customer supplies Example Corp with your ARN, it cannot control the external ID that Example Corp includes in its request to AWS. This helps prevent an unauthorized customer from gaining access to your resources.

The following diagram illustrates this.

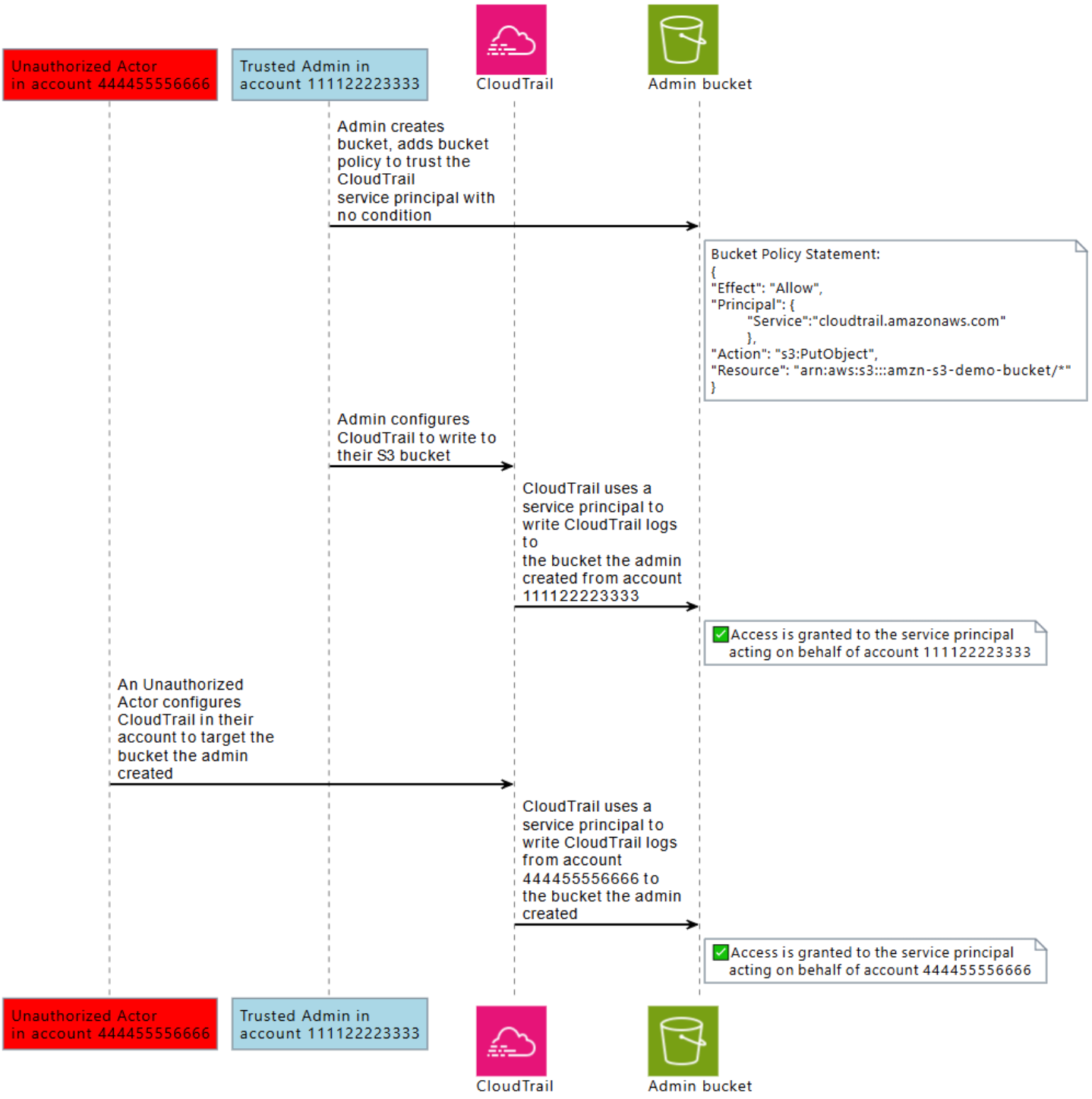


1. As before, when you start using Example Corp's service, you provide the ARN of **AWS1:ExampleRole** to Example Corp.
2. When Example Corp uses that role ARN to assume the role **AWS1:ExampleRole**, Example Corp includes your external ID (12345) in the AssumeRole API call. The external ID matches the role's trust policy, so the AssumeRole API call succeeds and Example Corp obtains temporary security credentials to access resources in your AWS account.
3. Another AWS customer also starts using Example Corp's service, and as before, this customer also provides the ARN of **AWS1:ExampleRole** for Example Corp to use.
4. But this time, when Example Corp attempts to assume the role **AWS1:ExampleRole**, it provides the external ID associated with the other customer (67890). The other customer has no way to change this. Example Corp does this because the request to use the role came from the other customer, so 67890 indicates the circumstance in which Example Corp is acting. Because you added a condition with your own external ID (12345) to the trust policy of **AWS1:ExampleRole**, the AssumeRole API call fails. The other customer is prevented from gaining unauthorized access to resources in your account (indicated by the red "X" in the diagram).

The external ID helps prevent any other customer from tricking Example Corp into unwittingly accessing your resources.

Cross-service confused deputy prevention

The following diagram demonstrates the cross-service confused deputy problem using the CloudTrail and Amazon S3 interaction example, where an unauthorized actor writes CloudTrail logs to an Amazon S3 bucket they are not authorized to have access to.



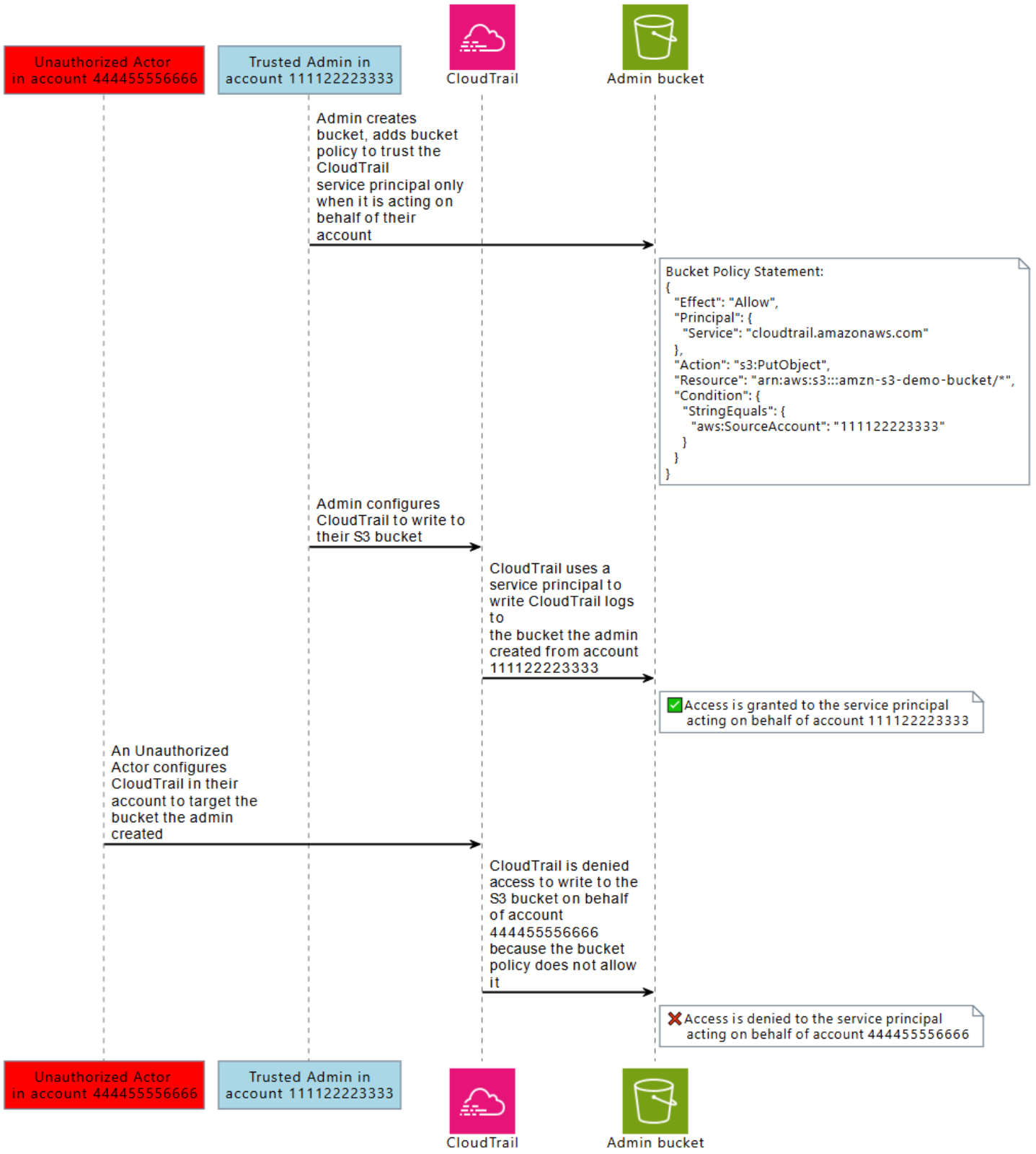
To help protect against an unauthorized actor from using the trust of an AWS principal to gain access to your resources, AWS service principals include information about the AWS resource, AWS account, and AWS organization they're acting on behalf of.

This information is available in global condition key values that can be used in a resource policy, or resource control policy for requests made by AWS service principals. We recommend using

aws:SourceArn, aws:SourceAccount, aws:SourceOrgID, or aws:SourceOrgPaths in your resource policies wherever an AWS service principal is granted permission to access one of your resources. These condition keys allow you to test in your resource policies, or resource control policies that AWS service principals accessing your resources are doing so on behalf of AWS resources, AWS accounts, or AWS Organizations you expect.

- Use `aws:SourceArn` to allow an AWS service principal to access your resources on behalf of a specific resource, such as a specific AWS CloudTrail trail or AppStream fleet.
- Use `aws:SourceAccount` to allow an AWS service principal to access your resources on behalf of a specific AWS account.
- Use `aws:SourceOrgID` to allow an AWS service principal to access your resources on behalf of specific AWS Organizations.
- Use `aws:SourceOrgPaths` to allow AWS service principal to access your resources on behalf of a specific AWS Organizations path.

The following diagram demonstrates the cross-service confused deputy scenario when a resource is configured with the `aws:SourceAccount` global condition context key, and an unauthorized actor from another account tries to access AWS resources they are not intended to have access to.



Using `aws:SourceArn`, `aws:SourceAccount`, `aws:SourceOrgID`, and `aws:SourceOrgPaths` global condition keys in a policy help you ensure service principals are accessing your resources on

your behalf. We recommend using these condition keys whenever access to one of your resources is granted to an AWS service principal.

> **Note**
>
> Some AWS service interactions have additional controls to help protect against cross-service confused deputy issues that test user access to a resource. For example, when a KMS key grant is issued to an AWS service, AWS KMS uses the encryption context associated with the resource, and the key grant to help protect against cross-service confused deputy issues.
> Please see the documentation of the services you use for more information about service-specific mechanisms that can help avoid cross-service confused deputy risks, and whether `aws:SourceArn`, `aws:SourceAccount`, `aws:SourceOrgID`, and `aws:SourceOrgPaths` are supported.

Cross-service confused deputy protection with resource-based policies

The following example policy grants the service principal `cloudtrail.amazonaws.com` access to the Amazon S3 bucket, arn:aws:s3:::amzn-s3-demo-bucket1, only when the service principal is acting on behalf of AWS account 111122223333.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CloudTrailAclCheck",
            "Effect": "Allow",
            "Principal": {"Service": "cloudtrail.amazonaws.com"},
            "Action": "s3:GetBucketAcl",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                }
            }
        },
        {
```

```
            "Sid": "AWSCloudTrailWrite",
            "Effect": "Allow",
            "Principal": {"Service": "cloudtrail.amazonaws.com"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1/[optionalPrefix]/Logs/
myAccountID/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

This example bucket policy grants the service principal `appstream.amazonaws.com` access to the powershell script examplefile.psh within the s3://amzn-s3-demo-bucket2 only when it is acting on behalf of the specified Amazon AppStream fleet, by specifying the fleet arn with `aws:SourceArn`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "appstream.amazonaws.com"
                ]
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket2/examplefile.psh",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:appstream:us-
east-1:111122223333:fleet/ExampleFleetName"
                }
            }
        }
    ]
```

```
}
```

Cross-service confused deputy protection with resource control policies

You can use resource control policies (RCP) to apply cross-service confused deputy controls to resources of supported AWS services. RCPs allow you to centrally apply cross-service confused deputy controls on your resources. You can use condition keys like `aws:SourceOrgId` and `aws:SourceOrgPaths` with RCPs attached to your AWS Organizations, organization units (OU) or AWS accounts in your organization without adding statements to specific resource-based policies. For more information about RCPs and supported services, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.

The following example RCP denies AWS service principals access to Amazon S3 buckets in your member accounts when `aws:SourceOrgID` is not equal to o-ExampleOrg. A corresponding allow must be present in the resource-based policy of the S3 bucket to allow AWS service principals with a `SourceOrgID` equal to o-ExampleOrg.

This policy applies the control only on requests by service principals (`"Bool": {"aws:PrincipalIsAWSService": "true"}`) that have the `aws:SourceAccount` key present (`"Null": {"aws:SourceAccount": "false"}`), so that service integrations that don't require the use of the condition key and calls by your principals aren't impacted. If the `aws:SourceAccount` condition key is present in the request context, the Null condition will evaluate to true, causing `aws:SourceOrgID` to be enforced. We use `aws:SourceAccount` instead of `aws:SourceOrgID` in the Null condition operator so that the control still applies if the request originates from an account that doesn't belong to an organization.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "RCPEnforceConfusedDeputyProtectionForS3",
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "s3:*"
      ],
      "Resource": "*",
```

```
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceOrgID": "o-ExampleOrg"
        },
        "Null": {
          "aws:SourceAccount": "false"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "true"
        }
      }
    }
  ]
}
```

Common scenarios for IAM roles

As with most AWS features, you generally have two ways to use a role: interactively in the IAM console, or programmatically with the AWS CLI, Tools for Windows PowerShell, or API.

- IAM users in your account using the IAM console can *switch to* a role to temporarily use the permissions of the role in the console. The users give up their original permissions and take on the permissions assigned to the role. When the users exit the role, their original permissions are restored.
- An application or a service offered by AWS (like Amazon EC2) can *assume* a role by requesting temporary security credentials for a role with which to make programmatic requests to AWS. You use a role this way so that you don't have to share or maintain long-term security credentials (for example, by creating an IAM user) for each entity that requires access to a resource.

> **Note**
>
> This guide uses the phrases *switch to a role* and *assume a role* interchangeably.

The simplest way to use roles is to grant your IAM users permissions to switch to roles that you create within your own or another AWS account. They can switch roles easily using the IAM console to use permissions that you don't ordinarily want them to have, and then exit the role to surrender those permissions. This can help prevent *accidental* access to or modification of sensitive resources.

For more complex uses of roles, such as granting access to applications and services, or federated external users, you can call the `AssumeRole` API. This API call returns a set of temporary credentials that the application can use in subsequent API calls. Actions attempted with the temporary credentials have only the permissions granted by the associated role. An application doesn't have to "exit" the role the way a user in the console does; rather the application simply stops using the temporary credentials and resumes making calls with the original credentials.

Federated users sign in by using credentials from an identity provider (IdP). AWS then provides temporary credentials to the trusted IdP to pass on to the user for including in subsequent AWS resource requests. Those credentials provide the permissions granted to the assigned role.

This section provides overviews of the following scenarios:

- Provide access for an IAM user in one AWS account that you own to access resources in another account that you own
- Provide access to non AWS workloads
- Provide access to IAM users in AWS accounts owned by third parties
- Provide access for services offered by AWS to AWS resources
- Provide access for externally authenticated users (identity federation)

Access for an IAM user in another AWS account that you own

You can grant your IAM users permission to switch to roles within your AWS account or to roles defined in other AWS accounts that you own.

> **Note**
>
> If you want to grant access to an account that you do not own or control, see Access to AWS accounts owned by third parties later in this topic.

Imagine that you have Amazon EC2 instances that are critical to your organization. Instead of directly granting your users permission to terminate the instances, you can create a role with those privileges. Then allow administrators to switch to the role when they need to terminate an instance. Doing this adds the following layers of protection to the instances:

- You must explicitly grant your users permission to assume the role.

- Your users must actively switch to the role using the AWS Management Console or assume the role using the AWS CLI or AWS API.
- You can add multi-factor authentication (MFA) protection to the role so that only users who sign in with an MFA device can assume the role. To learn how to configure a role so that users who assume the role must first be authenticated using multi-factor authentication (MFA), see Secure API access with MFA.

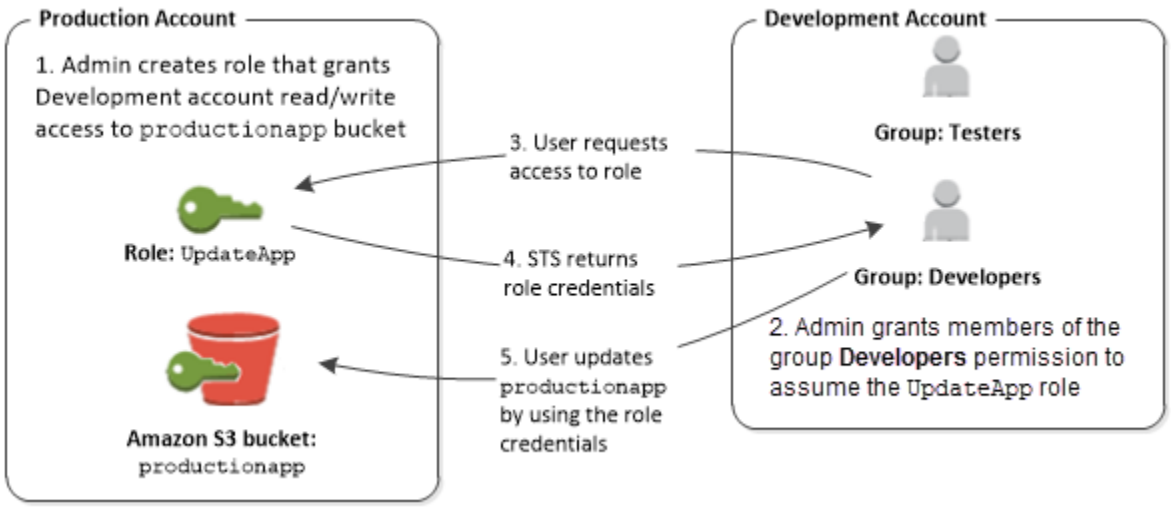
We recommend using this approach to enforce the *principle of least privilege*. That means restricting the use of elevated permissions to only those times when they are needed for specific tasks. With roles you can help prevent accidental changes to sensitive environments, especially if you combine them with auditing to help make sure that roles are only used when needed.

When you create a role for this purpose, you specify the accounts by ID whose users need access in the `Principal` element of the role's trust policy. You can then grant specific users in those other accounts permissions to switch to the role. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

A user in one account can switch to a role in the same or a different account. While using the role, the user can perform only the actions and access only the resources permitted by the role; their original user permissions are suspended. When the user exits the role, the original user permissions are restored.

Example scenario using separate development and production accounts

Imagine that your organization has multiple AWS accounts to isolate a development environment from a production environment. Users in the development account might occasionally need to access resources in the production account. For example, you might need cross-account access when you are promoting an update from the development environment to the production environment. Although you could create separate identities (and passwords) for users who work in both accounts, managing credentials for multiple accounts makes identity management difficult. In the following figure, all users are managed in the development account, but some developers require limited access to the production account. The development account has two groups: Testers and Developers, and each group has its own policy.



1. In the production account, an administrator uses IAM to create the `UpdateApp` role in that account. In the role, the administrator defines a trust policy that specifies the development account as a `Principal`, meaning that authorized users from the development account can use the `UpdateApp` role. The administrator also defines a permissions policy for the role that specifies the read and write permissions to the Amazon S3 bucket named `productionapp`.

   The administrator then shares the appropriate information with anyone who needs to assume the role. That information is the account number and name of the role (for AWS console users) or the Amazon Resource Name (ARN) (for AWS CLI or AWS API access). The role ARN might look like `arn:aws:iam::123456789012:role/UpdateApp`, where the role is named `UpdateApp` and the role was created in account number 123456789012.

   > **Note**
   >
   > The administrator can optionally configure the role so that users who assume the role must first be authenticated using multi-factor authentication (MFA). For more information, see Secure API access with MFA.

2. In the development account, an administrator grants members of the Developers group permission to switch to the role. This is done by granting the Developers group permission to call the AWS Security Token Service (AWS STS) `AssumeRole` API for the `UpdateApp` role. Any IAM user that belongs to the Developers group in the development account can now switch to the `UpdateApp` role in the production account. Other users who are not in the developer group

do not have permission to switch to the role and therefore cannot access the S3 bucket in the production account.

3. The user requests to switch to the role:
   - AWS console: The user chooses the account name on the navigation bar and chooses **Switch Role**. The user specifies the account ID (or alias) and role name. Alternatively, the user can choose a link sent in email by the administrator. The link takes the user to the **Switch Role** page with the details already filled in.
   - AWS API/AWS CLI: A user in the Developers group of the development account calls the `AssumeRole` function to obtain credentials for the `UpdateApp` role. The user specifies the ARN of the `UpdateApp` role as part of the call. If a user in the Testers group makes the same request, the request fails because Testers do not have permission to call `AssumeRole` for the `UpdateApp` role ARN.
4. AWS STS returns temporary credentials:
   - AWS console: AWS STS verifies the request with the role's trust policy to make sure that the request is from a trusted entity (which it is: the development account). After verification, AWS STS returns temporary security credentials to the AWS console.
   - API/CLI: AWS STS verifies the request against the role's trust policy to make sure that the request is from a trusted entity (which it is: the Development account). After verification, AWS STS returns temporary security credentials to the application.
5. The temporary credentials allow access to the AWS resource:
   - AWS console: The AWS console uses the temporary credentials on behalf of the user for all subsequent console actions, in this case, to read and write to the `productionapp` bucket. The console cannot access any other resource in the production account. When the user exits the role, the user's permissions revert to the original permissions held before switching to the role.
   - API/CLI: The application uses the temporary security credentials to update the `productionapp` bucket. With the temporary security credentials, the application can only read from and write to the `productionapp` bucket and cannot access any other resource in the Production account. The application does not have to exit the role, but instead stops using the temporary credentials and uses the original credentials in subsequent API calls.

Additional resources

For more information, see the following:

- IAM tutorial: Delegate access across AWS accounts using IAM roles

Access for non-AWS workloads

An IAM role is an object in AWS Identity and Access Management (IAM) that is assigned permissions. When you assume that role using an IAM identity or an identity from outside of AWS, it provides you with temporary security credentials for your role session. You might have workloads running in your data center or other infrastructure outside of AWS that must access your AWS resources. Instead of creating, distributing, and managing long-term access keys, you can use AWS Identity and Access Management Roles Anywhere (IAM Roles Anywhere) to authenticate your non-AWS workloads. IAM Roles Anywhere uses X.509 certificates from your certificate authority (CA) to authenticate identities and securely provide access to AWS services with the temporary credentials provided by an IAM role.

To use IAM Roles Anywhere

1. Set up a CA using AWS Private Certificate Authority or use a CA from your own PKI infrastructure.
2. After you have set up a CA, you create an object in IAM Roles Anywhere called a *trust anchor*. This anchor establishes trust between IAM Roles Anywhere and your CA for authentication.
3. You can then configure your existing IAM roles, or create new roles that trust the IAM Roles Anywhere service.
4. Authenticate your non-AWS workloads with IAM Roles Anywhere using the trust anchor. AWS grants the non-AWS workload temporary credentials to the IAM role that has access to your AWS resources.

Additional resources

The following resources can help you learn more about providing access to non-AWS workloads.

- For more information about configuring IAM Roles Anywhere, see What is AWS Identity and Access Management Roles Anywhere in the *IAM Roles Anywhere User Guide*.
- To learn how to set up public key infrastructure (PKI) for IAM Roles Anywhere, see IAM Roles Anywhere with an external certificate authority in the *AWS Security Blog*.

Access to AWS accounts owned by third parties

When third parties require access to your organization's AWS resources, you can use roles to delegate access to them. For example, a third party might provide a service for managing your AWS

resources. With IAM roles, you can grant these third parties access to your AWS resources without sharing your AWS security credentials. Instead, the third party can access your AWS resources by assuming a role that you create in your AWS account. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

Third parties must provide you with the following information for you to create a role that they can assume:

- **The third party's AWS account ID**. You specify their AWS account ID as the principal when you define the trust policy for the role.
- **An external ID to uniquely associate with the role**. The external ID can be any identifier that is known only by you and the third party. For example, you can use an invoice ID between you and the third party, but do not use something that can be guessed, like the name or phone number of the third party. You must specify this ID when you define the trust policy for the role. The third party must provide this ID when they assume the role.
- **The permissions that the third party requires to work with your AWS resources**. You must specify these permissions when defining the role's permission policy. This policy defines what actions they can take and what resources they can access.

After you create the role, you must provide the role's Amazon Resource Name (ARN) to the third party. They require your role's ARN in order to assume the role.

> **Important**
>
> When you grant third parties access to your AWS resources, they can access any resource that you specify in the policy. Their use of your resources is billed to you. Ensure that you limit their use of your resources appropriately.

External IDs for third party access

An external ID allows the user that is assuming the role to assert the circumstances in which they are operating. It also provides a way for the account owner to permit the role to be assumed only under specific circumstances. The primary function of the external ID is to address and prevent The confused deputy problem.

> **Important**
>
> AWS does not treat the external ID as a secret. After you create a secret like an access key pair or a password in AWS, you cannot view them again. The external ID for a role can be seen by anyone with permission to view the role.

When should I use an external ID?

Use an external ID in the following situations:

- You are an AWS account owner and you have configured a role for a third party that accesses other AWS accounts in addition to yours. You should ask the third party for an external ID that it includes when it assumes your role. Then you check for that external ID in your role's trust policy. Doing so ensures that the external party can assume your role only when it is acting on your behalf.
- You are in the position of assuming roles on behalf of different customers like Example Corp in our previous scenario. You should assign a unique external ID to each customer and instruct them to add the external ID to their role's trust policy. You must then ensure that you always include the correct external ID in your requests to assume roles.

  You probably already have a unique identifier for each of your customers, and this unique ID is sufficient for use as an external ID. The external ID is not a special value that you need to create explicitly, or track separately, just for this purpose.

  You should always specify the external ID in your `AssumeRole` API calls. In addition when a customer gives you a role ARN, test whether you can assume the role both with and without the correct external ID. If you can assume the role without the correct external ID, don't store the customer's role ARN in your system. Wait until your customer has updated the role trust policy to require the correct external ID. In this way you help your customers to do the right thing, which helps to keep both of you protected against the confused deputy problem.

Example scenario using an external ID

For example, let's say that you decide to hire a third-party company called Example Corp to monitor your AWS account and help optimize costs. In order to track your daily spending, Example Corp needs to access your AWS resources. Example Corp also monitors many other AWS accounts for other customers.

Do not give Example Corp access to an IAM user and its long-term credentials in your AWS account. Instead, use an IAM role and its temporary security credentials. An IAM role provides a mechanism to allow a third party to access your AWS resources without needing to share long-term credentials (such as an IAM user access key).

You can use an IAM role to establish a trusted relationship between your AWS account and the Example Corp account. After this relationship is established, a member of the Example Corp account can call the AWS Security Token Service AssumeRole API to obtain temporary security credentials. The Example Corp members can then use the credentials to access AWS resources in your account.

> **Note**
>
> For more information about the AssumeRole and other AWS API operations that you can call to obtain temporary security credentials, see Compare AWS STS credentials.

Here's a more detailed breakdown of this scenario:

1. You hire Example Corp, so they create a unique customer identifier for you. They provide you with this unique customer ID and their AWS account number. You need this information to create an IAM role in the next step.

   > **Note**
   >
   > Example Corp can use any string value they want for the ExternalId, as long as it is unique for each customer. It can be a customer account number or even a random string of characters, as long as no two customers have the same value. It is not intended to be a 'secret'. Example Corp must provide the ExternalId value to each customer. What is crucial is that it must be generated by Example Corp and ***not*** their customers to ensure each external ID is unique.

2. You sign in to AWS and create an IAM role that gives Example Corp access to your resources. Like any IAM role, the role has two policies, a permission policy and a trust policy. The role's trust policy specifies who can assume the role. In our sample scenario, the policy specifies the AWS account number of Example Corp as the `Principal`. This allows identities from that account to assume the role. In addition, you add a `Condition` element to the trust policy. This `Condition`

tests the `ExternalId` context key to ensure that it matches the unique customer ID from Example Corp. For example:

```
   "Principal": {"AWS": "Example Corp's AWS account ID"},
   "Condition": {"StringEquals": {"sts:ExternalId": "Unique ID Assigned by Example
Corp"}}
```

3. The permission policy for the role specifies what the role allows someone to do. For example, you could specify that the role allows someone to manage only your Amazon EC2 and Amazon RDS resources but not your IAM users or groups. In our sample scenario, you use the permission policy to give Example Corp read-only access to all of the resources in your account.
4. After you create the role, you provide the Amazon Resource Name (ARN) of the role to Example Corp.
5. When Example Corp needs to access your AWS resources, someone from the company calls the AWS `sts:AssumeRole` API. The call includes the ARN of the role to assume and the ExternalId parameter that corresponds to their customer ID.

If the request comes from someone using Example Corp's AWS account, and if the role ARN and the external ID are correct, the request succeeds. It then provides temporary security credentials that Example Corp can use to access the AWS resources that your role allows.

In other words, when a role policy includes an external ID, anyone who wants to assume the role must be a principal in the role and must include the correct external ID.

Key points for external IDs

- In a multi-tenant environment where you support multiple customers with different AWS accounts, we recommend using one external ID per AWS account. This ID should be a random string generated by the third party.
- To require that the third party provides an external ID when assuming a role, update the role's trust policy with the external ID of your choice.
- To provide an external ID when you assume a role, use the AWS CLI or AWS API to assume that role. For more information, see the STS AssumeRole API operation, or the STS assume-role CLI operation.
- The `ExternalId` value must have a minimum of 2 characters and a maximum of 1,224 characters. The value must be alphanumeric without white space. It can also include the

following symbols: plus (+), equal (=), comma (,), period (.), at (@), colon (:), forward slash (/), and hyphen (-).

Additional resources

The following resources can help you learn more about providing access to AWS accounts owned by third parties.

- To learn how to allow others to perform actions in your AWS account, see Create a role using custom trust policies.
- To learn how to grant permission to switch to a role, see Grant a user permissions to switch roles
- To learn how to create and provide trusted users with temporary security credentials, see Permissions for temporary security credentials.

Access to an AWS service

Many AWS services require that you use roles to control what that service can access. A role that a service assumes to perform actions on your behalf is called a service role. When a role serves a specialized purpose for a service, it can be categorized as a service-linked role. See the AWS documentation for each service to see if it uses roles and to learn how to assign a role for the service to use.

For details about creating a role to delegate access to a service offered by AWS, see Create a role to delegate permissions to an AWS service.

Access to externally authenticated users (identity federation)

Your users might already have identities outside of AWS, such as in your corporate directory. If those users need to work with AWS resources (or work with applications that access those resources), then those users also need AWS security credentials. You can use an IAM role to specify permissions for users whose identity is federated from your organization or a third-party identity provider (IdP).

> **Note**
>
> As a security best practice, we recommend you manage user access in IAM Identity Center with identity federation instead of creating IAM users. For information about specific situations where an IAM user is required, see When to create an IAM user (instead of a role).

IAM Identity Center permission sets provision IAM role templates that IAM Identity Center manages. Account access manager — an IAM feature — lets you assign your own custom IAM roles to IAM Identity Center users and groups, enabling capabilities like custom trust policies, role tags, and configurable role paths. You can use it alongside permission sets or on its own.

Federating users of a mobile or web-based app with Amazon Cognito

If you create a mobile or web-based app that accesses AWS resources, the app needs security credentials in order to make programmatic requests to AWS. For most mobile application scenarios, we recommend that you use Amazon Cognito. You can use this service with the AWS Mobile SDK for iOS and the AWS Mobile SDK for Android and Fire OS to create unique identities for users and authenticate them for secure access to your AWS resources. Amazon Cognito supports the same identity providers as those listed in the next section, and it also supports developer authenticated identities and unauthenticated (guest) access. Amazon Cognito also provides API operations for synchronizing user data so that it is preserved as users move between devices. For more information, see Amazon Cognito for mobile apps.

Federating users with public identity service providers or OpenID Connect

Whenever possible, use Amazon Cognito for mobile and web-based application scenarios. Amazon Cognito does most of the behind-the-scenes work with public identity provider services for you. It works with the same third-party services and also supports anonymous sign-ins. However, for more advanced scenarios, you can work directly with a third-party service like Login with Amazon, Facebook, Google, or any IdP that is compatible with OpenID Connect (OIDC). For more information about using OIDC federation using one of these services, see OIDC federation.

Federating users with SAML 2.0

If your organization already uses an identity provider software package that supports SAML 2.0 (Security Assertion Markup Language 2.0), you can create trust between your organization as an identity provider (IdP) and AWS as the service provider. You can then use SAML to provide your users with federated single-sign on (SSO) to the AWS Management Console or federated access to call AWS API operations. For example, if your company uses Microsoft Active Directory and Active Directory Federation Services, then you can federate using SAML 2.0. For more information about federating users with SAML 2.0, see SAML 2.0 federation.

Federating users by creating a custom identity broker application

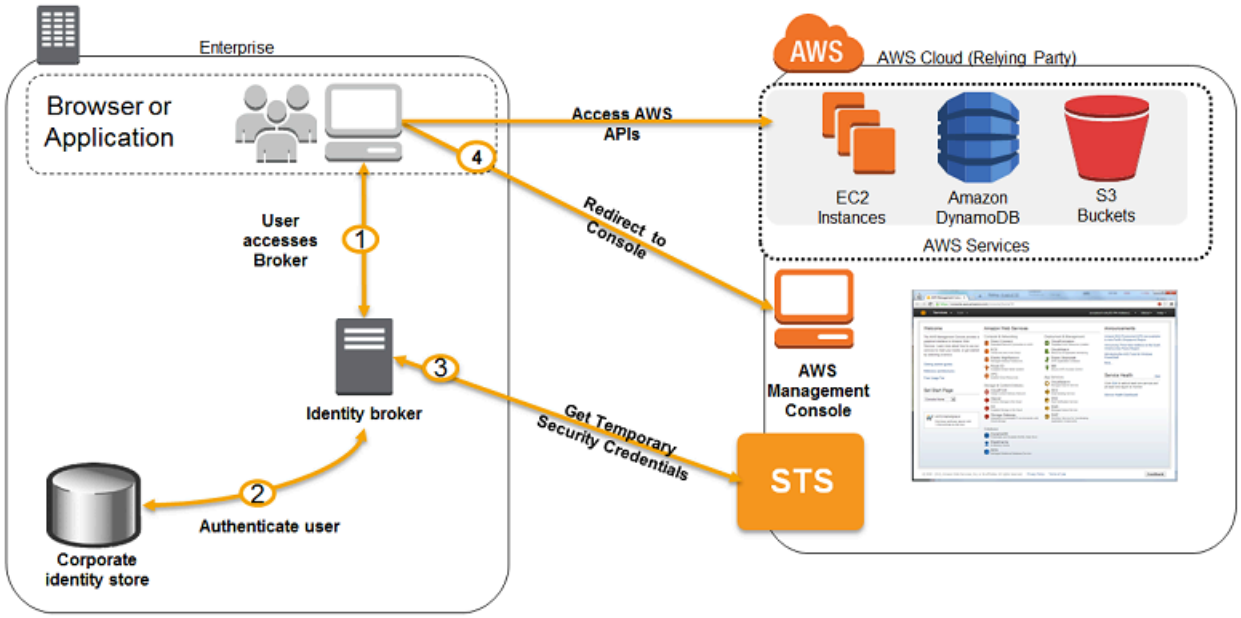
If your identity store is not compatible with SAML 2.0, then you can build a custom identity broker application to perform a similar function. The broker application authenticates users, requests

temporary credentials for users from AWS, and then provides them to the user to access AWS resources.

For example, Example Corp. has many employees who need to run internal applications that access the company's AWS resources. The employees already have identities in the company identity and authentication system, and Example Corp. doesn't want to create a separate IAM user for each company employee.

Bob is a developer at Example Corp. To enable Example Corp. internal applications to access the company's AWS resources, Bob develops a custom identity broker application. The application verifies that employees are signed into the existing Example Corp. identity and authentication system, which might use LDAP, Active Directory, or another system. The identity broker application then obtains temporary security credentials for the employees. This scenario is similar to the previous one (a mobile app that uses a custom authentication system), except that the applications that need access to AWS resources all run within the corporate network, and the company has an existing authentication system.

To get temporary security credentials, the identity broker application calls either `AssumeRole` or `GetFederationToken` to obtain temporary security credentials, depending on how Bob wants to manage the policies for users and when the temporary credentials should expire. (For more information about the differences between these API operations, see Temporary security credentials in IAM and Permissions for temporary security credentials.) The call returns temporary security credentials consisting of an AWS access key ID, a secret access key, and a session token. The identity broker application makes these temporary security credentials available to the internal company application. The app can then use the temporary credentials to make calls to AWS directly. The app caches the credentials until they expire, and then requests a new set of temporary credentials. The following figure illustrates this scenario.



This scenario has the following attributes:

- The identity broker application has permissions to access IAM's token service (STS) API to create temporary security credentials.
- The identity broker application is able to verify that employees are authenticated within the existing authentication system.
- Users are able to get a temporary URL that gives them access to the AWS Management Console (which is referred to as single sign-on).

For information about creating temporary security credentials, see Compare AWS STS credentials. For more information about SAML federated principals getting access to the AWS Management Console, see Enabling SAML 2.0 federated principals to access the AWS Management Console.

IAM role creation

To create a role, you can use the AWS Management Console, the AWS CLI, the Tools for Windows PowerShell, or the IAM API.

If you use the AWS Management Console, a wizard guides you through the steps for creating a role. The wizard has slightly different steps depending on whether you're creating a role for an AWS service, for an AWS account, or for a SAML or OIDC federated principal.

Roles for IAM users

Create this role to delegate permissions within your AWS account or to roles defined in other AWS accounts that you own. A user in one account can switch to a role in the same or a different account. While using the role, the user can perform only the actions and access only the resources permitted by the role; their original user permissions are suspended. When the user exits the role, the original user permissions are restored.

For more information, see Create a role to give permissions to an IAM user.

For more information about creating roles for cross account access, see Create a role using custom trust policies.

Roles for AWS services

Create this role to delegate permissions to a service that can perform actions on your behalf. A service role that you pass to a service must have an IAM policy with the permissions that allow the service to perform actions associated with that service. Different permissions are required for each AWS service.

For more information about creating service roles, see Create a role to delegate permissions to an AWS service.

For more information about creating service-linked roles, see Create a service-linked role.

Roles for identity federation

Create this role to delegate permissions to users that already have identities outside of AWS. When you use an identity provider, you don't have to create custom sign-in code or manage your own user identities. Your external users sign in through an IdP, and you can give those external identities permissions to use AWS resources in your account. Identity providers help keep your AWS account secure because you don't have to distribute or embed long-term security credentials, such as access keys, in your application.

For more information, see Create a role for a third-party identity provider.

Create a role to give permissions to an IAM user

You can use IAM roles to provide access to your AWS resources. With IAM roles, you can establish trust relationships between your *trusting* account and other AWS *trusted* accounts. The trusting account owns the resource to be accessed and the trusted account contains the users who

need access to the resource. However, it is possible for another account to own a resource in your account. For example, the trusting account might allow the trusted account to create new resources, such as creating new objects in an Amazon S3 bucket. In that case, the account that creates the resource owns the resource and controls who can access that resource.

After you create the trust relationship, an IAM user or an application from the trusted account can use the AWS Security Token Service (AWS STS) `AssumeRole` API operation. This operation provides temporary security credentials that enable access to AWS resources in your account.

The accounts can both be controlled by you, or the account with the users can be controlled by a third party. If the other account with the users is an AWS account that you do not control, then you can use the `externalId` attribute. The external ID can be any word or number that is agreed upon between you and the administrator of the third-party account. This option automatically adds a condition to the trust policy that allows the user to assume the role only if the request includes the correct `sts:ExternalID`. For more information, see Access to AWS accounts owned by third parties.

For information about how to use roles to delegate permissions, see Roles terms and concepts. For information about using a service role to allow services to access resources in your account, see Create a role to delegate permissions to an AWS service.

Creating an IAM role (console)

You can use the AWS Management Console to create a role that an IAM user can assume. For example, assume that your organization has multiple AWS accounts to isolate a development environment from a production environment. For high-level information about creating a role that allows users in the development account to access resources in the production account, see Example scenario using separate development and production accounts.

Minimum permissions

To perform the following steps, you must have at least the following IAM permissions:

- `access-analyzer:ValidatePolicy`
- `iam:AttachRolePolicy`
- `iam:CreatePolicy`
- `iam:CreateRole`
- `iam:GetAccountSummary`

- `iam:GetPolicy`
- `iam:GetPolicyVersion`
- `iam:GetRole`
- `iam:ListAccountAliases`
- `iam:ListAttachedRolePolicies`
- `iam:ListOpenIDConnectProviders`
- `iam:ListPolicies`
- `iam:ListRolePolicies`
- `iam:ListRoles`
- `iam:ListRoleTags`
- `iam:ListSAMLProviders`

Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. Choose **AWS account** role type.
4. To create a role for your account, choose **This account**. To create a role for another account, choose **Another AWS account** and enter the **Account ID** to which you want to grant access to your resources.

   The administrator of the specified account can grant permission to assume this role to any IAM user in that account. To do this, the administrator attaches a policy to the user or a group that grants permission for the `sts:AssumeRole` action. That policy must specify the role's ARN as the `Resource`.
5. If you are granting permissions to users from an account that you do not control, and the users will assume this role programmatically, select **Require external ID**. The external ID can be any word or number that is agreed upon between you and the administrator of the third party account. This option automatically adds a condition to the trust policy that allows the user to assume the role only if the request includes the correct `sts:ExternalID`. For more information, see Access to AWS accounts owned by third parties.

> **Important**
>
> Choosing this option restricts access to the role only through the AWS CLI, Tools for Windows PowerShell, or the AWS API. This is because you cannot use the AWS console to switch to a role that has an `externalId` condition in its trust policy. However, you can create this kind of access programmatically by writing a script or an application using the relevant SDK. For more information and a sample script, see How to Enable Cross-Account Access to the AWS Management Console in the AWS Security Blog.

6. If you want to restrict the role to users who sign in with multi-factor authentication (MFA), select **Require MFA**. This adds a condition to the role's trust policy that checks for an MFA sign-in. A user who wants to assume the role must sign in with a temporary one-time password from a configured MFA device. Users without MFA authentication cannot assume the role. For more information about MFA, see AWS Multi-factor authentication in IAM
7. Choose **Next**.
8. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions policy or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see Creating IAM policies. After you create the policy, close that tab and return to your original tab. Select the checkbox next to the permissions policies that you want anyone who assumes the role to have. If you prefer, you can select no policies at this time, and then attach policies to the role later. By default, a role has no permissions.
9. (Optional) Set a permissions boundary. This is an advanced feature.

   Open the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. Select the policy to use for the permissions boundary.
10. Choose **Next**.
11. For **Role name**, enter a name for your role. Role names must be unique within your AWS account. When a role name is used in a policy or as part of an ARN, the role name is case sensitive. When a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive. Because various entities might reference the role, you can't edit the name of the role after it is created.
12. (Optional) For **Description**, enter a description for the new role.

13. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the use cases and permissions for the role. You will be returned to previous pages to make the edits.
14. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
15. Review the role and then choose **Create role**.

> **Important**
>
> Remember that this is only the first half of the configuration required. You must also give individual users in the trusted account permissions to switch to the role in the console, or assume the role programmatically. For more information about this step, see Grant a user permissions to switch roles.

Creating an IAM role (AWS CLI)

Creating a role from the AWS CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the AWS CLI you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. Optionally, you can also set the permissions boundary for your role.

To create a role for cross-account access (AWS CLI)

1. Create a role: aws iam create-role
2. Attach a managed permissions policy to the role: aws iam attach-role-policy

   or

   Create an inline permissions policy for the role: aws iam put-role-policy
3. (Optional) Add custom attributes to the role by attaching tags: aws iam tag-role

   For more information, see Managing tags on IAM roles (AWS CLI or AWS API).
4. (Optional) Set the permissions boundary for the role: aws iam put-role-permissions-boundary

   A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

The following example shows the first two, and most common steps for creating a cross-account role in a simple environment. This example allows any user in the `123456789012` account to assume the role and view the `example_bucket` Amazon S3 bucket. This example also assumes that you are using a client computer running Windows, and have already configured your command line interface with your account credentials and Region. For more information, see Configuring the AWS Command Line Interface.

In this example, include the following trust policy in the first command when you create the role. This trust policy allows users in the `123456789012` account to assume the role using the `AssumeRole` operation, but only if the user provides MFA authentication using the `SerialNumber` and `TokenCode` parameters. For more information about MFA, see AWS Multi-factor authentication in IAM.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Principal": { "AWS": "arn:aws:iam::123456789012:root" },
          "Action": "sts:AssumeRole",
          "Condition": { "Bool": { "aws:MultiFactorAuthPresent": "true" } }
      }
  ]
}
```

Important

If your `Principal` element contains the ARN for a specific IAM role or user, then that ARN is transformed to a unique principal ID when the policy is saved. This helps mitigate the risk of someone escalating their permissions by removing and recreating the role or user. You don't normally see this ID in the console because there is also a reverse transformation back to the ARN when the trust policy is displayed. However, if you delete the role or user, then the principal ID appears in the console because AWS can no longer map it back to an ARN. Therefore, if you delete and recreate a user or role referenced in a trust policy's `Principal` element, you must edit the role to replace the ARN.

When you use the second command, you must attach an existing managed policy to the role. The following permissions policy allows anyone who assumes the role to perform only the `ListBucket` action on the `example_bucket` Amazon S3 bucket.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": "s3:ListBucket",
          "Resource": "arn:aws:s3:::example_bucket"
      }
  ]
}
```

To create this `Test-UserAccess-Role` role, you must first save the previous trust policy with the name `trustpolicyforacct123456789012.json` to the `policies` folder in your local `C:` drive. Then save the previous permissions policy as a customer managed policy in your AWS account with the name `PolicyForRole`. You can then use the following commands to create the role and attach the managed policy.

```
# Create the role and attach the trust policy file that allows users in the specified
 account to assume the role.
$ aws iam create-role --role-name Test-UserAccess-Role --assume-role-policy-document
 file://C:\policies\trustpolicyforacct123456789012.json

# Attach the permissions policy (in this example a managed policy) to the role to
 specify what it is allowed to do.
$ aws iam attach-role-policy --role-name Test-UserAccess-Role --policy-arn
 arn:aws:iam::123456789012:policy/PolicyForRole
```

Important

Remember that this is only the first half of the configuration required. You must also give individual users in the trusted account permissions to switch to the role. For more information about this step, see Grant a user permissions to switch roles.

After you create the role and grant it permissions to perform AWS tasks or access AWS resources, any users in the 123456789012 account can assume the role. For more information, see Switch to an IAM role (AWS CLI).

Creating an IAM role (AWS API)

Creating a role from the AWS API involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the API you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. Optionally, you can also set the permissions boundary for your role.

To create a role in code (AWS API)

1. Create a role: CreateRole

   For the role's trust policy, you can specify a file location.
2. Attach a managed permission policy to the role: AttachRolePolicy

   or

   Create an inline permission policy for the role: PutRolePolicy

   > **Important**
   >
   > Remember that this is only the first half of the configuration required. You must also give individual users in the trusted account permissions to switch to the role. For more information about this step, see Grant a user permissions to switch roles.
3. (Optional) Add custom attributes to the role by attaching tags: TagRole

   For more information, see Managing tags on IAM roles (AWS CLI or AWS API).
4. (Optional) Set the permissions boundary for the role: PutRolePermissionsBoundary

   A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

After you create the role and grant it permissions to perform AWS tasks or access AWS resources, you must grant permissions to users in the account to allow them to assume the role. For more information about assuming a role, see Switch to an IAM role (AWS API).

Creating an IAM role (AWS CloudFormation)

For information about creating an IAM role in AWS CloudFormation, see the resource and property reference and examples in the *AWS CloudFormation User Guide*.

For more information about IAM templates in AWS CloudFormation, see AWS Identity and Access Management template snippets in the *AWS CloudFormation User Guide*.

Create a role to delegate permissions to an AWS service

Many AWS services require that you use roles to allow the service to access resources in other services on your behalf. A role that a service assumes to perform actions on your behalf is called a service role. When a role serves a specialized purpose for a service, it is categorized as a service-linked role. To see what services support using service-linked roles, or whether a service supports any form of temporary credentials, see AWS services that work with IAM. To learn how an individual service uses roles, choose the service name in the table to view the documentation for that service.

When setting the `PassRole` permission, you should make sure that a user doesn't pass a role where the role has more permissions than you want the user to have. For example, Alice might not be allowed to perform any Amazon S3 actions. If Alice could pass a role to a service that allows Amazon S3 actions, the service could perform Amazon S3 actions on behalf of Alice when executing the job.

For information about how roles help you to delegate permissions, see Roles terms and concepts.

Service role permissions

You must configure permissions to allow an IAM entity (user or role) to create or edit a service role.

> **Note**
>
> The ARN for a service-linked role includes a service principal, which is indicated in the following policies as *`SERVICE-NAME`*`.amazonaws.com`. Do not try to guess the service principal, because it is case-sensitive and the format can vary across AWS services. To view the service principal for a service, see its service-linked role documentation.

To allow an IAM entity to create a specific service role

Add the following policy to the IAM entity that needs to create the service role. This policy allows you to create a service role for the specified service and with a specific name. You can then attach managed or inline policies to that role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreateRole",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/SERVICE-ROLE-NAME"
        }
    ]
}
```

To allow an IAM entity to create any service role

AWS recommends that you allow only administrative users to create any service role. A person with permissions to create a role and attach any policy can escalate their own permissions. Instead, create a policy that allows them to create only the roles that they need or have an administrator create the service role on their behalf.

To attach a policy that allows an administrator to access your entire AWS account, use the AdministratorAccess AWS managed policy.

To allow an IAM entity to edit a service role

Add the following policy to the IAM entity that needs to edit the service role.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "EditSpecificServiceRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DetachRolePolicy",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListRolePolicies",
                "iam:PutRolePolicy",
                "iam:UpdateRole",
                "iam:UpdateRoleDescription"
            ],
            "Resource": "arn:aws:iam::*:role/SERVICE-ROLE-NAME"
        },
        {
            "Sid": "ViewRolesAndPolicies",
            "Effect": "Allow",
            "Action": [
                "iam:GetPolicy",
                "iam:ListRoles"
            ],
            "Resource": "*"
        }
    ]
}
```

To allow an IAM entity to delete a specific service role

Add the following statement to the permissions policy for the IAM entity that needs to delete the specified service role.

```
{
    "Effect": "Allow",
    "Action": "iam:DeleteRole",
    "Resource": "arn:aws:iam::*:role/SERVICE-ROLE-NAME"
}
```

To allow an IAM entity to delete any service role

AWS recommends that you allow only administrative users to delete any service role. Instead, create a policy that allows them to delete only the roles that they need or have an administrator delete the service role on their behalf.

To attach a policy that allows an administrator to access your entire AWS account, use the AdministratorAccess AWS managed policy.

Creating a role for an AWS service (console)

You can use the AWS Management Console to create a role for a service. Because some services support more than one service role, see the AWS documentation for your service to see which use case to choose. You can learn how to assign the necessary trust and permissions policies to the role so that the service can assume the role on your behalf. The steps that you can use to control the permissions for your role can vary, depending on how the service defines the use cases, and whether or not you create a service-linked role.

Console

To create a role for an AWS service (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose a service, and then choose the use case. Use cases are defined by the service to include the trust policy that the service requires.
5. Choose **Next**.
6. For **Permissions policies**, the options depend on the use case that you selected:
   - If the service defines the permissions for the role, you can't select permissions policies.
   - Select from a limited set of permission polices.
   - Select from all permission policies.
   - Select no permissions policies, create the policies after the role is created, and then attach the policies to the role.
7. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

a. Open the **Set permissions boundary** section, and then choose **Use a permissions boundary to control the maximum role permissions**.

   IAM includes a list of the AWS managed and customer-managed policies in your account.

b. Select the policy to use for the permissions boundary.

8. Choose **Next**.
9. For **Role name**, the options depend on the service:

   - If the service defines the role name, you can't edit the role name.
   - If the service defines a prefix for the role name, you can enter an optional suffix.
   - If the service doesn't define the role name, you can name the role.

   > **Important**
   >
   > When you name a role, note the following:
   >
   > - Role names must be unique within your AWS account, and can't be made unique by case.
   >
   >   For example, don't create roles named both **`PRODROLE`** and **`prodrole`**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.
   > - You can't edit the name of the role after it's created because other entities might reference the role.

10. (Optional) For **Description**, enter a description for the role.
11. (Optional) To edit the use cases and permissions for the role, in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.
12. (Optional) To help identify, organize, or search for the role, add tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources in the *IAM User Guide*.
13. Review the role, and then choose **Create role**.

Creating a role for a service (AWS CLI)

Creating a role from the AWS CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the AWS CLI you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. If the service you are working with is Amazon EC2, then you must also create an instance profile and add the role to it. Optionally, you can also set the permissions boundary for your role.

To create a role for an AWS service from the AWS CLI

1. The following `create-role` command creates a role named *Test-Role* and attaches a trust policy to it:

   ```
   aws iam create-role --role-name Test-Role --assume-role-policy-document
   file://Test-Role-Trust-Policy.json
   ```

2. Attach a managed permissions policy to the role: aws iam attach-role-policy.

   For example, the following `attach-role-policy` command attaches the AWS managed policy named `ReadOnlyAccess` to the IAM role named `ReadOnlyRole`:

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
   ReadOnlyAccess --role-name ReadOnlyRole
   ```

   or

   Create an inline permissions policy for the role: aws iam put-role-policy

   To add an inline permissions policy, see the following example:

   ```
   aws iam put-role-policy --role-name Test-Role --policy-name
   ExamplePolicy --policy-document file://AdminPolicy.json
   ```

3. (Optional) Add custom attributes to the role by attaching tags: aws iam tag-role

   For more information, see Managing tags on IAM roles (AWS CLI or AWS API).

4. (Optional) Set the permissions boundary for the role: aws iam put-role-permissions-boundary

   A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

If you are going to use the role with Amazon EC2 or another AWS service that uses Amazon EC2, you must store the role in an instance profile. An instance profile is a container for a role that can be attached to an Amazon EC2 instance when launched. An instance profile can contain only one role, and that limit cannot be increased. If you create the role using the AWS Management Console, the instance profile is created for you with the same name as the role. For more information about instance profiles, see Use instance profiles. For information about how to launch an EC2 instance with a role, see Controlling Access to Amazon EC2 Resources in the *Amazon EC2 User Guide*.

To create an instance profile and store the role in it (AWS CLI)

1. Create an instance profile: aws iam create-instance-profile
2. Add the role to the instance profile: aws iam add-role-to-instance-profile

The AWS CLI example command set below demonstrates the first two steps for creating a role and attaching permissions. It also shows the two steps for creating an instance profile and adding the role to the profile. This example trust policy allows the Amazon EC2 service to assume the role and view the `example_bucket` Amazon S3 bucket. The example also assumes that you are running on a client computer running Windows and have already configured your command line interface with your account credentials and Region. For more information, see Configuring the AWS Command Line Interface.

In this example, include the following trust policy in the first command when you create the role. This trust policy allows the Amazon EC2 service to assume the role.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": "ec2.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }
}
```

When you use the second command, you must attach a permissions policy to the role. The following example permissions policy allows the role to perform only the `ListBucket` action on the `example_bucket` Amazon S3 bucket.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::example_bucket"
  }
}
```

To create this `Test-Role-for-EC2` role, you must first save the previous trust policy with the name `trustpolicyforec2.json` and the previous permissions policy with the name `permissionspolicyforec2.json` to the `policies` directory in your local `C:` drive. You can then use the following commands to create the role, attach the policy, create the instance profile, and add the role to the instance profile.

```
# Create the role and attach the trust policy that allows EC2 to assume this role.
$ aws iam create-role --role-name Test-Role-for-EC2 --assume-role-policy-document
 file://C:\policies\trustpolicyforec2.json

# Embed the permissions policy (in this example an inline policy) to the role to
 specify what it is allowed to do.
$ aws iam put-role-policy --role-name Test-Role-for-EC2 --policy-name Permissions-
Policy-For-Ec2 --policy-document file://C:\policies\permissionspolicyforec2.json

# Create the instance profile required by EC2 to contain the role
$ aws iam create-instance-profile --instance-profile-name EC2-ListBucket-S3

# Finally, add the role to the instance profile
$ aws iam add-role-to-instance-profile --instance-profile-name EC2-ListBucket-S3 --
role-name Test-Role-for-EC2
```

When you launch the EC2 instance, specify the instance profile name in the **Configure Instance Details** page if you use the AWS console. If you use the `aws ec2 run-instances` CLI command, specify the `--iam-instance-profile` parameter.

Creating a role for a service (AWS API)

Creating a role from the AWS API involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the API you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. If the service you are working with is Amazon EC2, then you must also create an instance profile and add the role to it. Optionally, you can also set the permissions boundary for your role.

To create a role for an AWS service (AWS API)

1. Create a role: CreateRole

   For the role's trust policy, you can specify a file location.
2. Attach a managed permissions policy to the role: AttachRolePolicy

   or

   Create an inline permissions policy for the role: PutRolePolicy
3. (Optional) Add custom attributes to the role by attaching tags: TagRole

   For more information, see Managing tags on IAM roles (AWS CLI or AWS API).
4. (Optional) Set the permissions boundary for the role: PutRolePermissionsBoundary

   A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

If you are going to use the role with Amazon EC2 or another AWS service that uses Amazon EC2, you must store the role in an instance profile. An instance profile is a container for a role. Each instance profile can contain only one role, and that limit cannot be increased. If you create the role in the AWS Management Console, the instance profile is created for you with the same name as the role. For more information about instance profiles, see Use instance profiles. For information about how to launch an Amazon EC2 instance with a role, see Controlling Access to Amazon EC2 Resources in the *Amazon EC2 User Guide*.

To create an instance profile and store the role in it (AWS API)

1. Create an instance profile: CreateInstanceProfile
2. Add the role to the instance profile: AddRoleToInstanceProfile

Create a service-linked role

A service-linked role is a unique type of IAM role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all the permissions that the service requires to call other AWS services on your behalf. The linked service also defines how you create, modify, and delete a service-linked role. A service might automatically create or delete the role. It might allow you to create, modify, or delete the role as part of a wizard or process in the service. Or it might require that you use IAM to create or delete the role. Regardless of the method, service-linked roles simplify the process of setting up a service because you don't have to manually add permissions for the service to complete actions on your behalf.

> **Note**
>
> Remember that service roles are different from service-linked roles. A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*. A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

The linked service defines the permissions of its service-linked roles, and unless defined otherwise, only that service can assume the roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

Before you can delete the roles, you must first delete their related resources. This helps prevent you from inadvertently removing permission to access the resources.

> **Tip**
>
> For information about which services support using service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions

You must configure permissions for an IAM entity (user or role) to allow the user or role to create or edit the service-linked role.

> **Note**
>
> The ARN for a service-linked role includes a service principal, which is indicated in the policies below as *SERVICE-NAME*.amazonaws.com. Do not try to guess the service principal, because it is case sensitive and the format can vary across AWS services. To view the service principal for a service, see its service-linked role documentation.

To allow an IAM entity to create a specific service-linked role

Add the following policy to the IAM entity that needs to create the service-linked role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/SERVICE-
NAME.amazonaws.com/SERVICE-LINKED-ROLE-NAME-PREFIX*",
            "Condition": {"StringLike": {"iam:AWSServiceName": "SERVICE-
NAME.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/SERVICE-
NAME.amazonaws.com/SERVICE-LINKED-ROLE-NAME-PREFIX*"
        }
    ]
}
```

To allow an IAM entity to create any service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to create a service-linked role, or any service role that includes the needed policies. This policy statement does not allow the IAM entity to attach a policy to the role.

```
{
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/*"
}
```

To allow an IAM entity to edit the description of any service roles

Add the following statement to the permissions policy for the IAM entity that needs to edit the description of a service-linked role, or any service role.

```
{
    "Effect": "Allow",
    "Action": "iam:UpdateRoleDescription",
    "Resource": "arn:aws:iam::*:role/aws-service-role/*"
}
```

To allow an IAM entity to delete a specific service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to delete the service-linked role.

```
{
    "Effect": "Allow",
    "Action": [
```

```
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/SERVICE-
NAME.amazonaws.com/SERVICE-LINKED-ROLE-NAME-PREFIX*"
}
```

To allow an IAM entity to delete any service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to delete a service-linked role, but not service role.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/*"
}
```

To allow an IAM entity to pass an existing role to the service

Some AWS services allow you to pass an existing role to the service, instead of creating a new service-linked role. To do this, a user must have permissions to *pass the role* to the service. Add the following statement to the permissions policy for the IAM entity that needs to pass a role. This policy statement also allows the entity to view a list of roles from which they can choose the role to pass. For more information, see Grant a user permissions to pass a role to an AWS service.

```
{
  "Sid": "PolicyStatementToAllowUserToListRoles",
  "Effect": "Allow",
  "Action": ["iam:ListRoles"],
  "Resource": "*"
},
{
  "Sid": "PolicyStatementToAllowUserToPassOneSpecificRole",
  "Effect": "Allow",
  "Action": [ "iam:PassRole" ],
  "Resource": "arn:aws:iam::account-id:role/my-role-for-XYZ"
}
```

Indirect permissions with service-linked roles

The permissions granted by a service-linked role can be indirectly transferred to other users and roles. When a service-linked role is used by an AWS service, that service-linked role can use its own permissions to call other AWS services. This means that users and roles with permissions to call a service that uses a service-linked role may have indirect access to services that can be accessed by that service-linked role.

For example, when you create an Amazon RDS DB instance, a service-linked role for RDS is automatically created if one does not already exist. This service-linked role allows RDS to call Amazon EC2, Amazon SNS, Amazon CloudWatch Logs, and Amazon Kinesis on your behalf. If you allow users and roles in your account to modify or create RDS databases, then they may be able to indirectly interact with Amazon EC2, Amazon SNS, Amazon CloudWatch Logs logs, and Amazon Kinesis resources by calling RDS, as RDS would use its service-linked role to access those resources.

Methods to create a service-linked role

The method that you use to create a service-linked role depends on the service. In some cases, you don't need to manually create a service-linked role. For example, when you complete a specific action (such as creating a resource) in the service, the service might create the service-linked role for you. Or if you were using a service before it began supporting service-linked roles, then the service might have automatically created the role in your account. To learn more, see A new role appeared in my AWS account.

In other cases, the service might support creating a service-linked role manually using the service console, API, or CLI. For information about which services support using service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. To learn whether the service supports creating the service-linked role, choose the **Yes** link to view the service-linked role documentation for that service.

If the service does not support creating the role, then you can use IAM to create the service-linked role.

> **Important**
>
> Service-linked roles count toward your IAM roles in an AWS account limit, but if you have reached your limit, you can still create service-linked roles in your account. Only service-linked roles can exceed the limit.

Creating a service-linked role (console)

Before you create a service-linked role in IAM, find out whether the linked service automatically creates service-linked roles, In addition, learn whether you can create the role from the service's console, API, or CLI.

To create a service-linked role (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**. Then, choose **Create role**.
3. Choose the **AWS Service** role type.
4. Choose the use case for your service. Use cases are defined by the service to include the trust policy required by the service. Then, choose **Next**.
5. Choose one or more permissions policies to attach to the role. Depending on the use case that you selected, the service might do any of the following:

   - Define the permissions used by the role.
   - Allow you to choose from a limited set of permissions.
   - Allow you to choose from any permissions.
   - Allow you to select no policies at this time, create the policies later, and then attach them to the role.

   Select the checkbox next to the policy that assigns the permissions that you want the role to have, and then choose **Next**.

   > **Note**
   >
   > The permissions that you specify are available to any entity that uses the role. By default, a role has no permissions.

6. For **Role name**, the degree of role name customization is defined by the service. If the service defines the role's name, then this option is not editable. In other cases, the service might define a prefix for the role and let you enter an optional suffix.

   If possible, enter a role name suffix to add to the default name. This suffix helps you identify the purpose of this role. Role names must be unique within your AWS account. They are

not distinguished by case. For example, you cannot create roles named both **<service-linked-role-name>_SAMPLE** and **<service-linked-role-name>_sample**. Because various entities might reference the role, you cannot edit the name of the role after it has been created.

7. (Optional) For **Description**, edit the description for the new service-linked role.
8. You cannot attach tags to service-linked roles during creation. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
9. Review the role and then choose **Create role**.

Creating a service-linked role (AWS CLI)

Before creating a service-linked role in IAM, find out whether the linked service automatically creates service-linked roles and whether you can create the role from the service's CLI. If the service CLI is not supported, you can use IAM commands to create a service-linked role with the trust policy and inline policies that the service needs to assume the role.

To create a service-linked role (AWS CLI)

Run the following command:

```
aws iam create-service-linked-role --aws-service-name SERVICE-NAME.amazonaws.com
```

Creating a service-linked role (AWS API)

Before creating a service-linked role in IAM, find out whether the linked service automatically creates service-linked roles and whether you can create the role from the service's API. If the service API is not supported, you can use the AWS API to create a service-linked role with the trust policy and inline policies that the service needs to assume the role.

To create a service-linked role (AWS API)

Use the CreateServiceLinkedRole API call. In the request, specify a service name of *SERVICE_NAME_URL*`.amazonaws.com`.

For example, to create the **Lex Bots** service-linked role, use `lex.amazonaws.com`.

Create a role for a third-party identity provider

You can use identity providers instead of creating IAM users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user

identities permissions to access AWS resources in your account. For more information about federation and identity providers, see Identity providers and federation into AWS.

Creating a role for OIDC and SAML federated principals (console)

The procedures for creating a role depends on your choice of third party providers:

- For OpenID Connect (OIDC), see Create a role for OpenID Connect federation (console).
- For SAML 2.0, see Create a role for SAML 2.0 federation (console).

Creating a role for federated access (AWS CLI)

The steps to create a role for the supported identity providers (OIDC or SAML) from the AWS CLI are identical. The difference is in the contents of the trust policy that you create in the prerequisite steps. Begin by following the steps in the **Prerequisites** section for the type of provider you are using:

- For an OIDC provider, see Prerequisites for creating a role for OIDC.
- For a SAML provider, see Prerequisites for creating a role for SAML.

Creating a role from the AWS CLI involves multiple steps. When you use the console to create a role, many of the steps are done for you, but with the AWS CLI you must explicitly perform each step yourself. You must create the role and then assign a permissions policy to the role. Optionally, you can also set the permissions boundary for your role.

To create a role (AWS CLI)

1. Create a role: aws iam create-role
2. Attach a permissions policy to the role: aws iam attach-role-policy

   or

   Create an inline permissions policy for the role: aws iam put-role-policy
3. (Optional) Add custom attributes to the role by attaching tags: aws iam tag-role

   For more information, see Managing tags on IAM roles (AWS CLI or AWS API).
4. (Optional) Set the permissions boundary for the role: aws iam put-role-permissions-boundary

A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

The following example shows the first two, and most common, steps for creating an identity provider role in a simple environment. This example allows any user in the `123456789012` account to assume the role and view the `example_bucket` Amazon S3 bucket. This example also assumes that you are running the AWS CLI on a computer running Windows, and have already configured the AWS CLI with your credentials. For more information, see Configuring the AWS Command Line Interface.

The following example trust policy is designed for a mobile app if the user signs in using Amazon Cognito. In this example, *`us-east:12345678-ffff-ffff-ffff-123456`* represents the identity pool ID assigned by Amazon Cognito.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "RoleForCognito",
        "Effect": "Allow",
        "Principal": {"Federated": "cognito-identity.amazonaws.com"},
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {"StringEquals": {"cognito-identity.amazonaws.com:aud": "us-
east:12345678-ffff-ffff-ffff-123456"}}
    }
}
```

The following permissions policy allows anyone who assumes the role to perform only the `ListBucket` action on the `example_bucket` Amazon S3 bucket.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
```

```
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::example_bucket"
  }
}
```

To create this `Test-Cognito-Role` role, you must first save the previous trust policy with the name `trustpolicyforcognitofederation.json` and the previous permissions policy with the name `permspolicyforcognitofederation.json` to the `policies` folder in your local `C:` drive. You can then use the following commands to create the role and attach the inline policy.

```
# Create the role and attach the trust policy that enables users in an account to
 assume the role.
$ aws iam create-role --role-name Test-Cognito-Role --assume-role-policy-document
 file://C:\policies\trustpolicyforcognitofederation.json

# Attach the permissions policy to the role to specify what it is allowed to do.
aws iam put-role-policy --role-name Test-Cognito-Role --policy-name
 Perms-Policy-For-CognitoFederation --policy-document file://C:\policies
\permspolicyforcognitofederation.json
```

Creating a role for federated access (AWS API)

The steps to create a role for the supported identity providers (OIDC or SAML) from the AWS CLI are identical. The difference is in the contents of the trust policy that you create in the prerequisite steps. Begin by following the steps in the **Prerequisites** section for the type of provider you are using:

- For an OIDC provider, see Prerequisites for creating a role for OIDC.
- For a SAML provider, see Prerequisites for creating a role for SAML.

To create a role (AWS API)

1. Create a role: CreateRole
2. Attach a permissions policy to the role:AttachRolePolicy

   or

   Create an inline permissions policy for the role: PutRolePolicy
3. (Optional) Add custom attributes to the role by attaching tags: TagRole

For more information, see Managing tags on IAM roles (AWS CLI or AWS API).

4. (Optional) Set the permissions boundary for the role: PutRolePermissionsBoundary

   A permissions boundary controls the maximum permissions that a role can have. Permissions boundaries are an advanced AWS feature.

Create a role for OpenID Connect federation (console)

You can use OpenID Connect (OIDC) federated identity providers instead of creating AWS Identity and Access Management users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user identities permissions to access AWS resources in your account. For more information about federation and IdPs, see Identity providers and federation into AWS.

Prerequisites for creating a role for OIDC

Before you can create a role for OIDC federation, you must first complete the following prerequisite steps.

To prepare to create a role for OIDC federation

1. Sign up with one or more services offering federated OIDC identity. If you are creating an app that needs access to your AWS resources, you also configure your app with the provider information. When you do, the provider gives you an application or audience ID that is unique to your app. (Different providers use different terminology for this process. This guide uses the term *configure* for the process of identifying your app with the provider.) You can configure multiple apps with each provider, or multiple providers with a single app. View information about using the identity providers as follows:

   - Login with Amazon Developer Center
   - Add Facebook Login to Your App or Website on the Facebook developers site.
   - Using OAuth 2.0 for Login (OpenID Connect) on the Google developers site.

2. After you receive the required information from the IdP, create an IdP in IAM. For more information, see Create an OpenID Connect (OIDC) identity provider in IAM.

> **Important**
>
> If you are using an OIDC IdP from Google, Facebook, or Amazon Cognito, don't create a separate IAM IdP in the AWS Management Console. These OIDC identity providers are already built into AWS and are available for you to use. Skip this step and create new roles using your IdP in the following step.

3. Prepare the policies for the role that the IdP-authenticated users will assume. As with any role, a role for a mobile app includes two policies. One is the trust policy that specifies who can assume the role. The other is the permissions policy that specifies the AWS actions and resources that the mobile app is allowed or denied access to.

   For web IdPs, we recommend that you use Amazon Cognito to manage identities. In this case, use a trust policy similar to this example.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": {
           "Effect": "Allow",
           "Principal": {"Federated": "cognito-identity.amazonaws.com"},
           "Action": "sts:AssumeRoleWithWebIdentity",
           "Condition": {
               "StringEquals": {"cognito-identity.amazonaws.com:aud": "us-
   east-2:12345678-abcd-abcd-abcd-123456"},
               "ForAnyValue:StringLike": {"cognito-identity.amazonaws.com:amr":
    "unauthenticated"}
           }
       }
   }
   ```

   Replace `us-east-2:12345678-abcd-abcd-abcd-123456` with the identity pool ID that Amazon Cognito assigns to you.

   If you manually configure an OIDC IdP, when you create the trust policy, you must use three values that ensure that only your app can assume the role:

- For the `Action` element, use the `sts:AssumeRoleWithWebIdentity` action.
- For the `Principal` element, use the string `{"Federated":`*`providerUrl/providerArn`*`}`.
  - For some common OIDC IdPs, the *`providerUrl`* is a URL. The following examples include methods to specify the principal for some common IdPs:

    ```
    "Principal":{"Federated":"cognito-identity.amazonaws.com"}
    ```

    ```
    "Principal":{"Federated":"www.amazon.com"}
    ```

    ```
    "Principal":{"Federated":"graph.facebook.com"}
    ```

    ```
    "Principal":{"Federated":"accounts.google.com"}
    ```

  - For other OIDC providers, use the Amazon Resource Name (ARN) of the OIDC IdP that you created in Step 2, such as the following example:

    ```
    "Principal":{"Federated":"arn:aws:iam::123456789012:oidc-provider/
    server.example.com"}
    ```

- For the `Condition` element, use a `StringEquals` condition to limit permissions. Test the identity pool ID for Amazon Cognito) or the app ID for other providers. The identity pool ID should match the app ID that you received when you configured the app with the IdP. This match between the IDs ensures that the request comes from your app.

> **Note**
>
> IAM roles for Amazon Cognito identity pools trust the service principal `cognito-identity.amazonaws.com` to assume the role. Roles of this type must contain at least one condition key to limit the principals who can assume the role. Additional considerations apply to Amazon Cognito identity pools that assume cross-account IAM roles. The trust policies of these roles must accept the `cognito-identity.amazonaws.com` service principal and must contain the `aud` condition key to restrict role assumption to users from your intended identity pools. A policy that trusts Amazon Cognito identity pools without this condition creates a risk that a user from an unintended identity pool can assume the role. For more information, see Trust policies for IAM roles in Basic (Classic) authentication in the *Amazon Cognito Developer Guide*.

Create a condition element similar to one of the following examples, depending on the IdP that you are using:

```
"Condition": {"StringEquals": {"cognito-identity.amazonaws.com:aud":
"us-east:12345678-ffff-ffff-ffff-123456"}}
```

```
"Condition": {"StringEquals": {"www.amazon.com:app_id":
"amzn1.application-oa2-123456"}}
```

```
"Condition": {"StringEquals": {"graph.facebook.com:app_id":
"111222333444555"}}
```

```
"Condition": {"StringEquals": {"accounts.google.com:aud":
"66677788899900pro0"}}
```

For OIDC providers, use the fully qualified URL of the OIDC IdP with the aud context key, such as the following example:

```
"Condition": {"StringEquals": {"server.example.com:aud":
"appid_from_oidc_idp"}}
```

> **Note**
>
> The values for the principal in the trust policy for the role are specific to an IdP. A role for OIDC can specify only one principal. Therefore, if the mobile app allows users to sign in from more than one IdP, create a separate role for each IdP that you want to support. Create separate trust policies for each IdP.

If a user uses a mobile app to sign in from Login with Amazon, the following example trust policy would apply. In the example, *amzn1.application-oa2-123456* represents the app ID that Amazon assigns when you configured the app using Login with Amazon.

JSON

```
{
      "Version":"2012-10-17",
```

```
      "Statement": [{
          "Sid": "RoleForLoginWithAmazon",
          "Effect": "Allow",
          "Principal": {"Federated": "www.amazon.com"},
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {"StringEquals": {"www.amazon.com:app_id":
 "amzn1.application-oa2-123456"}}
      }]
  }
```

If a user uses a mobile app to sign in from Facebook, the following example trust policy would apply. In this example, *111222333444555* represents the app ID that Facebook assigns.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [{
          "Sid": "RoleForFacebook",
          "Effect": "Allow",
          "Principal": {"Federated": "graph.facebook.com"},
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {"StringEquals": {"graph.facebook.com:app_id":
 "111222333444555"}}
      }]
  }
```

If a user uses a mobile app to sign in from Google, the following example trust policy would apply. In this example, *666777888999000* represents the app ID that Google assigns.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [{
          "Sid": "RoleForGoogle",
          "Effect": "Allow",
```

```
          "Principal": {"Federated": "accounts.google.com"},
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {"StringEquals": {"accounts.google.com:aud":
 "666777888999000"}}
      }]
  }
```

If a user uses a mobile app to sign in from Amazon Cognito, the following example trust policy would apply. In this example, *us-east:12345678-ffff-ffff-ffff-123456* represents the identity pool ID that Amazon Cognito assigns.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [{
          "Sid": "RoleForCognito",
          "Effect": "Allow",
          "Principal": {"Federated": "cognito-identity.amazonaws.com"},
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {"StringEquals": {"cognito-
identity.amazonaws.com:aud": "us-east:12345678-ffff-ffff-ffff-123456"}}
      }]
  }
```

Creating a role for OIDC

After you complete the prerequisites, you can create the role in IAM. For recognized shared OpenID Connect (OIDC) identity providers (IdPs), IAM requires explicit evaluation of specific claims in JSON Web Tokens (JWTs) known as *identity-provider controls*. For more information about which OIDC IdPs have *identity-provider controls*, see Identity-provider controls for shared OIDC providers.

The following procedure describes how to create the role for OIDC federation in the AWS Management Console. To create a role from the AWS CLI or AWS API, see the procedures at Create a role for a third-party identity provider.

Important

If you use Amazon Cognito, use the Amazon Cognito console to set up the roles. Otherwise, use the IAM console to create a role for OIDC federation.

To create an IAM role for OIDC federation

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Choose **Web identity** as the trusted entity type and select **Next**.
4. For **Identity provider**, choose the IdP for your role:

   - If you want to create a role for an individual web IdP, choose **Login with Amazon**, **Facebook**, or **Google**.

     Note

     You must create a separate role for each IdP that you want to support.

   - If you want to create an advanced scenario role for Amazon Cognito, choose **Amazon Cognito**.

     Note

     You must manually create a role to use with Amazon Cognito only when you work on an advanced scenario. Otherwise, Amazon Cognito can create roles for you. For more information about Amazon Cognito, see Identity pools (federated identities) external identity providers in the *Amazon Cognito Developer Guide*.

   - If you want to create a role for GitHub Actions, you must start by adding the GitHub OIDC provider to IAM. After you've added the GitHub OIDC provider to IAM, choose **token.actions.githubusercontent.com**.

> **Note**
>
> For information about how to configure AWS to trust GitHub's OIDC provider as a federated identity, see GitHub Docs - Configuring OpenID Connect in Amazon Web Services. For information about best practices for limiting access for roles associated with the IAM IdP for GitHub, see Configuring a role for GitHub OIDC identity provider on this page.

- If you want to create a role for HashiCorp Cloud Platform (HCP) Terraform, you must start by adding the Terraform OIDC provider to IAM. After you've added the Terraform OIDC provider to IAM, choose **app.terraform.io**.

> **Important**
>
> IAM roles for HashiCorp Cloud Platform (HCP) Terraform OIDC provider must evaluate the IAM condition key, `app.terraform.io:sub`, in the role trust policy. This condition key limits which HCP Terraform organizations, projects, workspaces, or run phases are able to assume the role. Without this condition key, your trust policy grants access to your role and AWS resources by identities outside of your organization, which does not align with the principle of least privilege.
> If you set or modify a role trust policy for a role associated with the HCP Terraform OIDC provider in your AWS account, but do not evaluate IAM condition key `app.terraform.io:sub`, you will receive an error. Additionally, AWS STS will deny authorization requests if your role trust policy doesn't evaluate this condition key.

5. The requested information varies based on the OIDC provider you choose.

- Enter the identifier for your application. The label of the identifier changes based on the provider you choose:
  - If you want to create a role for Login with Amazon, enter the app ID into the **Application ID** box.
  - If you want to create a role for Facebook, enter the app ID into the **Application ID** box.
  - If you want to create a role for Google, enter the audience name into the **Audience** box.
  - If you want to create a role for Amazon Cognito, enter the ID of the identity pool that you have created for your Amazon Cognito applications into the **Identity Pool ID** box.
- If you want to create a role for GitHub Actions, enter the following details:

- For **Audience**, choose `sts.amazonaws.com`.
- For **GitHub organization**, enter the GitHub organization name. The GitHub organization name is required and must be alphanumeric including dashes (-). You can't use wildcard characters (* and ?) in the GitHub organization name.
- (Optional) For **GitHub repository**, enter the GitHub repository name. If you don't specify a value, it defaults to a wildcard (*).
- (Optional) For **GitHub branch**, enter the GitHub branch name. If you don't specify a value, it defaults to a wildcard (*).

- If you want to create a role for HashiCorp Cloud Platform (HCP) Terraform, enter the following details:
  - For **Audience**, choose `aws.workload.identity`.
  - For **Organization**, enter the organization name. You can specify a wildcard character (*) for all organizations.
  - For **Project**, enter the project name. You can specify a wildcard character (*) for all projects.
  - For **Workspace**, enter the workspace name. You can specify a wildcard character (*) for all workspaces.
  - For **Run Phase**, enter the run phase name. You can specify a wildcard character (*) for all run phases.

6. (Optional) For **Condition (optional)**, choose **Add Condition** to create additional conditions that must be met before users of your application can use the permissions that the role grants. For example, you can add a condition that grants access to AWS resources only for a specific IAM user ID. You can also add conditions to the trust policy after the role is created. For more information, see Update a role trust policy.
7. Review your OIDC information and then choose **Next**.
8. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions policy, or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see Creating IAM policies. After you create the policy, close that tab and return to your original tab. Select the checkbox next to the permissions policies that you want OIDC users to have. If you prefer, you can select no policies at this time, and then attach policies to the role later. By default, a role has no permissions.
9. (Optional) Set a permissions boundary. This is an advanced feature.

Open the **Permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. Select the policy to use for the permissions boundary.

10. Choose **Next**.
11. For **Role name**, enter a role name. Role names must be unique within your AWS account. They are not case dependent. For example, you can't create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you can't edit the name of the role after you create it.
12. (Optional) For **Description**, enter a description for the new role.
13. To edit the use cases and permissions for the role, choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections.
14. (Optional) To add metadata to the role, attach tags as key–value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
15. Review the role and then choose **Create role**.

Configuring a role for GitHub OIDC identity provider

If you use GitHub as an OpenID Connect (OIDC) identity provider (IdP), best practice is to limit the entities that can assume the role associated with the IAM IdP. When you include a condition statement in the trust policy, you can limit the role to a specific GitHub organization, repository, or branch. You can use the condition key `token.actions.githubusercontent.com:sub` with string condition operators to limit access. We recommend that you limit the condition to a specific set of repositories or branches within your GitHub organization. For information about how to configure AWS to trust GitHub's OIDC as a federated identity, see GitHub Docs - Configuring OpenID Connect in Amazon Web Services.

If you use GitHub environments in action workflows or in OIDC policies, we strongly recommend adding protection rules to the environment for additional security. Use deployment branches and tags to restrict which branches and tags can deploy to the environment. For more information on configuring environments with protection rules, see Deployment branches and tags in GitHub's *Using environments for deployment* article.

When GitHub's OIDC IdP is the trusted Principal for your role, IAM checks the role trust policy condition to verify that the condition key `token.actions.githubusercontent.com:sub` is present and that its value is not solely a wildcard character (* and ?) or null. IAM performs this check when the trust policy is created or updated. If the condition key

`token.actions.githubusercontent.com:sub` is not present, or the key value doesn't satisfy the mentioned value criteria, the request will fail and return an error.

> **Important**
>
> If you do not limit the condition key `token.actions.githubusercontent.com:sub` to a specific organization or repository, then GitHub Actions from organizations or repositories outside of your control are able to assume roles associated with the GitHub IAM IdP in your AWS account.

The following example trust policy limits access to the defined GitHub organization, repository, and branch. The condition key `token.actions.githubusercontent.com:sub` value in the following example is the default subject value format documented by GitHub.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::012345678910:oidc-provider/
token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
          "token.actions.githubusercontent.com:sub":
 "repo:GitHubOrg/GitHubRepo:ref:refs/heads/GitHubBranch"
        }
      }
    }
  ]
}
```

The following example condition limits access to the defined GitHub organization and repository, but grants access to any branch within the repository.

```
"Condition": {
  "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
  },
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "repo:GitHubOrg/GitHubRepo:*"
  }
}
```

The following example condition limits access to any repository or branch within the defined GitHub organization. We recommend that you limit the condition key `token.actions.githubusercontent.com:sub` to a specific value that limits access to GitHub Actions from within your GitHub organization.

```
"Condition": {
  "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
  },
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "repo:GitHubOrg/*"
  }
}
```

For more information about the OIDC federation keys available for condition checks in policies, see Available keys for AWS OIDC federation.

Create a role for SAML 2.0 federation (console)

You can use SAML 2.0 federation instead of creating IAM users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user identities permissions to access AWS resources in your account. For more information about federation and identity providers, see Identity providers and federation into AWS.

> **Note**
>
> To improve federation resiliency, we recommend that you configure your IdP and AWS federation to support multiple SAML sign-in endpoints. For details, see the AWS Security Blog article How to use regional SAML endpoints for failover.

Prerequisites for creating a role for SAML

Before you can create a role for SAML 2.0 federation, you must first complete the following prerequisite steps.

To prepare to create a role for SAML 2.0 federation

1. Before you create a role for SAML-based federation, you must create a SAML provider in IAM. For more information, see Create a SAML identity provider in IAM.
2. Prepare the policies for the role that the SAML 2.0–authenticated users will assume. As with any role, a role for the SAML federation includes two policies. One is the role trust policy that specifies who can assume the role. The other is the IAM permissions policy that specifies the AWS actions and resources that the SAML federated principal is allowed or denied access to.

   When you create the trust policy for your role, you must use three values to ensure that only your application can assume the role:

   - For the `Action` element, use the `sts:AssumeRoleWithSAML` action.
   - For the `Principal` element, use the string `{"Federated":`*`ARNofIdentityProvider`*`}`. Replace *`ARNofIdentityProvider`* with the ARN of the SAML identity provider that you created in Step 1.
   - For the `Condition` element, use a `StringEquals` condition to test that the `saml:aud` attribute from the SAML response matches the URL your browser displays when signing into the console. This sign-in endpoint URL is your identity provider's SAML recipient attribute. You can include sign-in URLs within particular regions. AWS recommends using Regional endpoints instead of the global endpoint to improve federation resiliency. For a list of possible *`region-code`* values, see the **Region** column in AWS Sign-In endpoints.

     If SAML encryption is required, the sign-in URL must include the unique identifier AWS assigns to your SAML provider. You can view the unique identifier by selecting the identity provider in the IAM console to display the details page.

```
https://region-code.signin.aws.amazon.com/saml/acs/IdP-ID
```

The following example trust policy is designed for a SAML federated user:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRoleWithSAML",
        "Principal": {
            "Federated": "arn:aws:iam::111122223333:saml-provider/PROVIDER-
NAME"
        },
        "Condition": {
            "StringEquals": {
                "SAML:aud": "https://region-code.signin.aws.amazon.com/saml"
            }
        }
    }
}
```

Replace the principal ARN with the actual ARN for the SAML provider that you created in IAM. It will have your own account ID and provider name.

Creating a role for SAML

After you complete the prerequisite steps, you can create the role for SAML-based federation.

To create a role for SAML-based federation

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles** and then choose **Create role**.
3. Choose the **SAML 2.0 federation** role type.
4. For **Select a SAML provider**, choose the provider for your role.

5. Choose the SAML 2.0 access level method.

   - Choose **Allow programmatic access only** to create a role that can be assumed programmatically from the AWS API or AWS CLI.
   - Choose **Allow programmatic and AWS Management Console access** to create a role that can be assumed programmatically and from the AWS Management Console.

   The roles created by both are similar, but the role that can also be assumed from the console includes a trust policy with a particular condition. That condition explicitly ensures that the SAML audience (`SAML:aud` attribute) is set to the AWS sign-in endpoint for your SAML provider.

6. The procedure for defining attributes varies depending on the access type.

   - If you're creating a role for programmatic access, choose an attribute from the **Attribute** list. Then, in the **Value** box, enter a value to include in the role. This restricts role access to users from the identity provider whose SAML authentication response (assertion) includes the attributes that you specify. You must specify at least one attribute to ensure that your role is limited to a subset of users at your organization.
   - If you're creating a role for programmatic and AWS Management Console access, the **Sign-in endpoints** section defines the URL your browser displays when signing into the console. This endpoint is your identity provider's SAML recipient attribute, which maps to the `saml:aud` context key. For more information, see Configure SAML assertions for the authentication response.

     a. Choose **Regional endpoints** or **Non-regional endpoint**. We recommend using multiple Regional SAML sign-in endpoints to improve federation resiliency.

     b. For **Regions**, choose the regions that your SAML provider supports for AWS sign-in.

     c. For **Sign-in URLs to include unique identifiers**, select whether the sign-in endpoints include the unique identifiers AWS assigns to your SAML identity provider. This option is required for encrypted SAML assertions. For more information, see SAML 2.0 federation.

7. To add more attribute-related conditions to the trust policy, choose **Condition (optional)**, select the additional condition, and specify a value.

> **Note**
>
> The list includes the most commonly used SAML attributes. IAM supports additional attributes that you can use to create conditions. For a list of the supported attributes, see Available Keys for SAML Federation. If you need a condition for a supported SAML attribute that's not in the list, you can manually add that condition. To do that, edit the trust policy after you create the role.

8. Review your SAML 2.0 trust information and then choose **Next**.
9. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions policy, or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see Creating IAM policies. After you create the policy, close that tab and return to your original tab. Select the checkbox next to the permissions policies that you want SAML federated users to have. If you prefer, you can select no policies at this time, and then attach policies to the role later. By default, a role has no permissions.
10. (Optional) Set a permissions boundary. This is an advanced feature.

    Open the **Permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. Select the policy to use for the permissions boundary.
11. Choose **Next**.
12. Choose **Next: Review**.
13. For **Role name**, enter a role name. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.
14. (Optional) For **Description**, enter a description for the new role.
15. Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the use cases and permissions for the role.
16. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
17. Review the role and then choose **Create role**.

After you create the role, you complete the SAML trust by configuring your identity provider software with information about AWS. This information includes the roles that you want your SAML federated users to use. This is referred to as configuring the relying party trust between your IdP and AWS. For more information, see Configure your SAML 2.0 IdP with relying party trust and adding claims.

Create a role using custom trust policies

You can create a custom trust policy to delegate access and allow others to perform actions in your AWS account. For more information, see Creating IAM policies.

For information about how to use roles to delegate permissions, see Roles terms and concepts.

Creating an IAM role using a custom trust policy (console)

You can use the AWS Management Console to create a role that an IAM user can assume. For example, assume that your organization has multiple AWS accounts to isolate a development environment from a production environment. For high-level information about creating a role that allows users in the development account to access resources in the production account, see Example scenario using separate development and production accounts.

To create a role using a custom trust policy (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. Choose the **Custom trust policy** role type.
4. In the **Custom trust policy** section, enter or paste the custom trust policy for the role. For more information, see Creating IAM policies.
5. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.
6. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

   Open the **Permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions boundary.
7. Choose **Next**.

8. For **Role name**, the degree of role name customization is defined by the service. If the service defines the role's name, this option is not editable. In other cases, the service might define a prefix for the role and allow you to enter an optional suffix. Some services allow you to specify the entire name of your role.

   If possible, enter a role name or role name suffix. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **`PRODROLE`** and **`prodrole`**. Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.
9. (Optional) For **Description**, enter a description for the new role.
10. (Optional) Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the custom policy and permissions for the role.
11. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
12. Review the role and then choose **Create role**.

Examples of policies for delegating access

The following examples show how you can allow or grant an AWS account access to the resources in another AWS account. To learn how to create an IAM policy using these example JSON policy documents, see the section called “Creating policies using the JSON editor”.

Topics

- Using roles to delegate access to the resources of another AWS account resources
- Using a policy to delegate access to services
- Using a resource-based policy to delegate access to an Amazon S3 bucket in another account
- Using a resource-based policy to delegate access to an Amazon SQS queue in another account
- Cannot delegate access when the account is denied access

Using roles to delegate access to the resources of another AWS account resources

For a tutorial that shows how to use IAM roles to grant users in one account access to AWS resources that are in another account, see IAM tutorial: Delegate access across AWS accounts using IAM roles.

Important

You can include the ARN for a specific role or user in the `Principal` element of a role trust policy. When you save the policy, AWS transforms the ARN to a unique principal ID. This helps mitigate the risk of someone escalating their privileges by removing and recreating the role or user. You don't normally see this ID in the console, because there is also a reverse transformation back to the ARN when the trust policy is displayed. However, if you delete the role or user, then the relationship is broken. The policy no longer applies, even if you recreate the user or role because it does not match the principal ID stored in the trust policy. When this happens, the principal ID shows up in the console because AWS can no longer map it back to an ARN. The result is that if you delete and recreate a user or role referenced in a trust policy's `Principal` element, you must edit the role to replace the ARN. It is transformed into the new principal ID when you save the policy.

Using a policy to delegate access to services

The following example shows a policy that can be attached to a role. The policy enables two services, Amazon EMR and AWS Data Pipeline, to assume the role. The services can then perform any tasks granted by the permissions policy assigned to the role (not shown). To specify multiple service principals, you do not specify two `Service` elements; you can have only one. Instead, you use an array of multiple service principals as the value of a single `Service` element.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "elasticmapreduce.amazonaws.com",
          "datapipeline.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

Using a resource-based policy to delegate access to an Amazon S3 bucket in another account

In this example, account A uses a resource-based policy (an Amazon S3 bucket policy) to grant account B full access to account A's S3 bucket. Then account B creates an IAM user policy to delegate that access to account A's bucket to one of the users in account B.

The S3 bucket policy in account A might look like the following policy. In this example, account A's S3 bucket is named *amzn-s3-demo-bucket*, and account B's account number is 111122223333. It does not specify any individual users or groups in account B, only the account itself.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "AccountBAccess1",
    "Effect": "Allow",
    "Principal": {"AWS": "111122223333"},
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ]
  }
}
```

Alternatively, account A can use Amazon S3 Access Control Lists (ACLs) to grant account B access to an S3 bucket or a single object within a bucket. In that case, the only thing that changes is how account A grants access to account B. Account B still uses a policy to delegate access to an IAM group in account B, as described in the next part of this example. For more information about controlling access on S3 buckets and objects, see Access Control in the *Amazon Simple Storage Service User Guide*.

The administrator of account B might create the following policy sample. The policy allows read access to a group or user in account B. The preceding policy grants access to account B. However, individual groups and users in account B cannot access the resource until a group or user policy

explicitly grants permissions to the resource. The permissions in this policy can only be a subset of those in the preceding cross-account policy. Account B cannot grant more permissions to its groups and users than account A granted to account B in the first policy. In this policy, the `Action` element is explicitly defined to allow only `List` actions, and the `Resource` element of this policy matches the `Resource` for the bucket policy implemented by account A.

To implement this policy account B uses IAM to attach it to the appropriate user (or group) in account B.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:List*",
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ]
  }
}
```

Using a resource-based policy to delegate access to an Amazon SQS queue in another account

In the following example, account A has an Amazon SQS queue that uses a resource-based policy attached to the queue to grant queue access to account B. Then account B uses an IAM group policy to delegate access to a group in account B.

The following example queue policy gives account B permission to perform the `SendMessage` and `ReceiveMessage` actions on account A's queue named *queue1*, but only between noon and 3:00 p.m. on November 30, 2014. Account B's account number is 1111-2222-3333. Account A uses Amazon SQS to implement this policy.

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "111122223333"},
    "Action": [
      "sqs:SendMessage",
      "sqs:ReceiveMessage"
    ],
    "Resource": ["arn:aws:sqs:*:123456789012:queue1"],
    "Condition": {
      "DateGreaterThan": {"aws:CurrentTime": "2014-11-30T12:00Z"},
      "DateLessThan": {"aws:CurrentTime": "2014-11-30T15:00Z"}
    }
  }
}
```

Account B's policy for delegating access to a group in account B might look like the following example. Account B uses IAM to attach this policy to a group (or user).

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:queue1"
  }
}
```

In the preceding IAM user policy example, account B uses a wildcard to grant its user access to all Amazon SQS actions on account A's queue. However account B can delegate access only to the extent that account B has been granted access. The account B group that has the second policy can access the queue only between noon and 3:00 p.m. on November 30, 2014. The user can only perform the `SendMessage` and `ReceiveMessage` actions, as defined in account A's Amazon SQS queue policy.

Cannot delegate access when the account is denied access

An AWS account cannot delegate access to another account's resources if the other account has explicitly denied access to the user's parent account. The deny propagates to the users under that account whether or not the users have existing policies granting them access.

For example, account A writes a bucket policy on account A's S3 bucket that explicitly denies account B access to account A's bucket. But account B writes an IAM user policy that grants a user in account B access to account A's bucket. The explicit deny applied to account A's S3 bucket propagates to the users in account B. It overrides the IAM user policy granting access to the user in account B. (For detailed information how permissions are evaluated, see Policy evaluation logic.)

Account A's bucket policy might look like the following policy. In this example, account A's S3 bucket is named *amzn-s3-demo-bucket*, and account B's account number is 1111-2222-3333. Account A uses Amazon S3 to implement this policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "AccountBDeny",
    "Effect": "Deny",
    "Principal": {"AWS": "111122223333"},
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
  }
}
```

This explicit deny overrides any policies in account B that provide permission to access the S3 bucket in account A.

Create roles automatically with role manager

Many AWS services depend on IAM roles to perform actions on your behalf. IAM role manager is an optional account setting that automatically provisions such roles, so you don't have to set them up yourself. Role manager is designed to help you get started building with AWS services without additional manual IAM configuration.

Role manager is available in select AWS service consoles, listed in the section called “AWS service consoles that support role manager” on this page.

How role manager works

When role manager is enabled, it provides roles appropriate to the service and the resource as part of the relevant console workflow. Supported service consoles will indicate when role manager is available. No additional action is required to accept the role that role manager provides. For example, when you create an AWS Lambda function in the console, Lambda uses role manager to create the function's execution role by default. To use a different role, choose an existing role or manually create a new one.

Role manager works by applying a role template, a definition of a role's trust policy and permissions for a service and use case. To learn more about the template that each service uses, see Overview of role templates. A template for a defined task grants only the actions that task needs. A template for open-ended work, such as running your own code, grants broad access to AWS APIs so you can build without granting access to each service yourself.

IAM roles created by role manager behave like any other role you create. You can view, edit, or delete them. Each role also records the role template it came from, so you can identify the roles that role manager created. AWS CloudTrail logs when role manager creates roles so you can audit when and how each role was created. Role manager role creation appears in CloudTrail as the AcquireRole event.

Prerequisites

Enabling role manager requires the `iam:PutAccountProperties` and `iam:CreateServiceLinkedRole` permissions. Disabling role manager requires the `iam:PutAccountProperties` permission. Viewing the current setting requires the `iam:GetAccountProperties` permission. The AWS managed policy `IAMFullAccess` includes all of these permissions. For all the permissions that role manager requires, see Manage access to role manager.

How to enable and disable role manager

You can enable or disable role manager using the AWS Management Console, AWS CLI, or AWS API. Role manager is controlled by the account-level property `RoleManager/Enabled`, which you set using the `PutAccountProperties` API. A value of `true` enables role manager; `false` disables it.

> **Note**
>
> For most AWS accounts, role manager is disabled by default. But if you created your account using our new AWS experience, role manager is enabled by default and cannot be disabled until you activate advanced features. For more information, see Activate advanced AWS features.

To enable or disable role manager (console)

Role manager may be enabled or disabled on the **Account settings** page in the IAM console. That page also shows whether role manager is enabled for the account.

1. Sign in to the AWS Management Console and open the IAM console.
2. In the navigation pane, choose **Account settings**.
3. In the **role manager** section, choose **Enable** or **Disable**.

Roles created by role manager are not deleted when you disable role manager.

To enable or disable role manager (AWS CLI)

Set the `RoleManager/Enabled` property to `true` to enable role manager, or `false` to disable it.

- To enable role manager, run `put-account-properties`:

```
aws iam put-account-properties --properties RoleManager/Enabled=true
```

- To disable role manager, run `put-account-properties`:

```
aws iam put-account-properties --properties RoleManager/Enabled=false
```

- To view whether role manager is enabled for the account, run `get-account-properties`:

```
aws iam get-account-properties
```

To enable or disable role manager (AWS API)

- To enable or disable role manager, call PutAccountProperties with the `RoleManager/Enabled` property set to `true` or `false`.

- To view the current setting, call GetAccountProperties.

For more information, see PutAccountProperties and GetAccountProperties in the *IAM API Reference*.

AWS service consoles that support role manager

Role manager supports role creation for the following AWS service consoles:

- AWS Backup
- Amazon Bedrock
- AWS CloudFormation
- Amazon CloudWatch
- Amazon Cognito
- Amazon EKS
- AWS Elastic Beanstalk
- Amazon EventBridge
- AWS Glue
- AWS Lambda
- Amazon RDS
- Amazon SageMaker Unified Studio
- AWS Secrets Manager
- AWS Step Functions

AWS will continue adding service support over time. To learn more about the role template that each service uses, see Overview of role templates.

Related information

- Manage access to role manager
- Apply least-privilege permissions to a role created automatically
- Overview of role templates

Manage access to role manager

You control access to role manager at two levels. IAM permissions determine who in an account can enable role manager and who can use the roles it provides. In an organization, service control policies (SCPs) determine what member accounts can do.

Permissions for role manager

To enable or disable role manager, you need the `iam:PutAccountProperties` permission. The AWS managed policy `IAMFullAccess` includes it.

To use role manager, a user needs the IAM permissions for the actions that role manager performs on their behalf. Role manager provides roles through the `AcquireRole` API, which evaluates each underlying IAM action against the user's permissions:

- If a role that matches the template already exists in the account, `AcquireRole` returns that role. This requires the `iam:GetRole` permission.
- If no matching role exists, `AcquireRole` creates one from the template. This requires the `iam:CreateRole` permission, plus the permissions for what the template defines: `iam:PutRolePolicy` if the template adds inline policies, and `iam:AttachRolePolicy` if the template attaches managed policies.

A user with full IAM access has all the permissions that role manager requires. To see the exact actions a specific template requires, retrieve the template with `GetRoleTemplateVersion` and review its policies.

When role manager creates a role, AWS CloudTrail records the creation as a single `AcquireRole` event. The event shows who called the operation, the role template and parameter values that were used, and the role that was created.

The following example shows that role manager called the `AcquireRole` operation on behalf of a user who assumed the role `Admin`, using the `PowerUserRoleTemplate` role template to create the role `PowerUserRole`.


```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123EXAMPLE:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
```

```
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "accountId": "123456789012",
        "accessKeyId": "ASIAWZYMLEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2026-08-11T18:18:50Z",
                "mfaAuthenticated": "false"
            }
        },
        "onBehalfOf": {
            "userId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "identityStoreArn": "arn:aws:identitystore::111122223333:identitystore/
a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
        }
    },
    "eventTime": "2026-08-11T18:23:00Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "AcquireRole",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/151.0.0.0 Safari/537.36",
    "requestParameters": {
        "templateArn": "arn:aws:iam::aws:role-template/iam.amazonaws.com/
PowerUserRoleTemplate:1",
        "templateMinorVersion": 0,
        "replacementValues": {
            "AWSServiceName": {
                "values": [
                    "iam.amazonaws.com"
                ]
            }
        }
    },
    "responseElements": {
        "role": {
```

```
            "path": "/",
            "roleName": "PowerUserRole",
            "roleId": "AROA987EXAMPLE",
            "arn": "arn:aws:iam::123456789012:role/PowerUserRole",
            "createDate": "2026-08-11T18:23:00Z",
            "assumeRolePolicyDocument": "%7B%22Version%22:%222012-10-17%22,
%22Statement%22:%5B%7B%22Effect%22:%22Allow%22,%22Principal%22:%7B%22Service%22:
%22iam.amazonaws.com%22%7D,%22Action%22:%22sts:AssumeRole%22%7D%5D%7D"
        }
    },
    "requestID": "fde3dac0-73ac-491c-876b-EXAMPLE89c57",
    "eventID": "c93332cf-1429-4aee-bcaa-EXAMPLE3790d",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "iam.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

Control role manager in an organization

You can control role manager across an organization with an AWS Organizations service control policy (SCP). The following examples show how to block enabling role manager, block role creation, or block template-based role creation only.

These examples use the deny list strategy, which means you also need `FullAWSAccess` or another policy that allows actions attached to your organization entities. You still need to grant permissions to your principals with identity-based policies.

To prevent member accounts from enabling role manager through the console or the API, deny the `iam:PutAccountProperties` permission. We recommend this policy when you create new accounts directly through Organizations.

```
{
    "Sid": "DenyRoleManagerEnablement",
    "Effect": "Deny",
    "Action": "iam:PutAccountProperties",
```

```
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "iam:AccountPropertyNamespaces": "RoleManager"
        }
    }
}
```

Block access to use role manager

To block the use of role manager, use the `iam:RoleTemplateARN` context key, which is present on the IAM actions that create a role from a template. The following policy denies all IAM actions that have a non-null value for the `iam:RoleTemplateARN` context key. This policy doesn't prevent all role creation—it blocks only role creation through role manager and template-based role creation through the SDK.

```
{
    "Sid": "DenyTemplatedRoleCreation",
    "Effect": "Deny",
    "Action": "iam:*",
    "Resource": "*",
    "Condition": {
        "Null": {
            "iam:RoleTemplateARN": false
        }
    }
}
```

Block role creation except through role manager

To allow role creation only through role manager, use the following policy to block non-role manager role creation. This policy still allows template-based role creation directly through the SDK.

```
{
    "Sid": "DenyNonTemplatedRoleCreation",
    "Effect": "Deny",
    "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy",
        "iam:PutRolePolicy",
        "iam:PutRolePermissionsBoundary",
```

```
            "iam:TagRole"
        ],
        "Resource": "*",
        "Condition": {
            "Null": {
                "iam:RoleTemplateARN": true
            }
        }
    }
```

Related information

- Create roles automatically with role manager
- Apply least-privilege permissions to a role created automatically
- Overview of role templates

Apply least-privilege permissions to a role created automatically

When creating a resource in a service console, role manager provides the IAM role that the resource needs. For some tasks, such as running your own code, AWS cannot determine in advance what permissions are required. As a result, a role can allow more permissions than the resource requires. After you know which permissions the resource needs, you can reduce the role's permissions to match.

Disabling role manager does not affect roles previously created by role manager. You can disable role manager without reducing any permissions, and you can reduce a role's permissions without disabling role manager.

When to apply least-privilege permissions

Granting a role only the permissions its resource requires is a security best practice in line with the principle of least privilege. For most roles, role manager creates roles with well scoped permissions. For some roles, such as those for compute resources or cloud infrastructure management, it creates roles with broad permissions.

When you are ready to start scoping down roles created by role manager, we recommend starting with the roles associated with your most sensitive workloads and resources, such as anything handling PII or production data. To learn more, see Prepare for least-privilege permissions.

How to analyze unused access

Before disabling role manager, AWS recommends that you use IAM Access Analyzer to create an unused access analyzer in your account. The analyzer compares the actions that a role is allowed to perform with the actions that it has performed, and reports the unused permissions. The analyzer reviews the IAM roles in your account, including roles that you created yourself.

Where the findings appear depends on how the analyzer was provided:

- **Analyzer provided by role manager:** When AWS provides the unused access analyzer for your account (see the note in this section), the findings appear on the **Roles** page in the IAM console. For each role, you can see the number of unused permissions and open a recommendation. Use the following procedure to reduce a role's permissions from the **Roles** page. After you disable role manager, the **Account settings** page provides a **View roles** button that opens the **Roles** page, where each role's findings appear.
- **Analyzer created by yourself:** When you create your own unused access analyzer, the findings and recommendations appear in the IAM Access Analyzer console. That experience is unchanged by role manager. To review and act on unused access findings, see IAM Access Analyzer findings.

> **Note**
>
> If AWS enabled role manager for your account created using the new AWS experience, you don't need to create an analyzer the first time you disable role manager. After you activate advanced features and disable role manager, AWS provides an unused access analyzer at no additional cost for 90 days. The analyzer gives you visibility into unused role permissions and policy scope-down recommendations so you can update the roles in your account, including those that role manager created, toward least privilege.

Prerequisites

To review Access Analyzer findings, you need permission to perform the `access-analyzer:ListFindings` action. To apply a recommendation, you need permission to perform `iam:CreatePolicy`, `iam:AttachRolePolicy`, and `iam:DetachRolePolicy` on the role.

To reduce a role's permissions in the console (analyzer provided by role manager)

These steps apply when AWS provided the unused access analyzer for your account. If you created your own analyzer, review and apply recommendations in the IAM Access Analyzer console instead. See IAM Access Analyzer findings.

1. Sign in to the AWS Management Console and open the IAM console.
2. In the navigation pane, choose **Roles**.
3. On the **Roles** page, in the **Findings** column, choose the **Unused permissions** link for the role whose permissions you want to reduce.
4. On the **Reduce role permissions** page, review the unused permissions reported for the role.
5. Compare the recommended permissions with the role's current permissions, and review the actions that the recommendation removes.
6. Choose **Make changes**.

How recommended permissions work

Based on the unused access analyzer findings, IAM creates one or more customer-managed policies that contain the recommended permissions. IAM attaches the new policies before it detaches the old ones, so the role keeps its permissions throughout the change.

For example, consider an AWS Lambda execution role. IAM Access Analyzer reports which of the role's permissions are unused and recommends a policy scoped to the actions that the role performed, such as writing logs to Amazon CloudWatch Logs. You review the recommendation against what the function does, confirm that it covers everything the function needs, and apply it. The role then allows only the actions the function needs.

> **Important**
>
> The recommendation is based on the last 30 days of activity. A permission that the resource uses infrequently on a longer time horizon, such as for a quarterly job, can appear as unused. Review the recommendation against what your workload needs before you apply it.
>
> After you change a role's policies or trust policies, role manager no longer manages that role. The role will not pick up any version changes to the role template that was originally used to create the role.

IAM limits the number of managed policies that you can attach to a role. If applying the recommendation would exceed that limit, the console notifies you and does not apply the change. For information about this quota, see IAM and AWS STS quotas.

Related information

- Create roles automatically with role manager
- Manage access to role manager
- Overview of role templates

Overview of role templates

A role template is a role blueprint used to create new roles. The template sets the role's trust policy and permissions so you get a ready-to-use role for a service without writing the policies yourself.

Role manager uses AWS-managed role templates, but you can also use role templates without role manager. This page explains how role templates work and lists the template for each service.

How role templates work

A role template contains a trust policy and permissions, and has an Amazon Resource Name (ARN). The trust policy defines which principals can assume the role. The permissions define what the role can do. Template versions are immutable. When AWS updates a template, it publishes a new version rather than changing an existing one.

Role templates may accept additional parameters to customize and scope the role's properties and permissions. For example, a template might take a `RoleName` for the role it creates, or a resource identifier such as a `bucketName`. When role manager creates a role in the console, it applies the parameters to the permissions and trust policy for the role. When you create a role from a template directly with the IAM API, you supply these parameters.

To create a role from a template, use the `iam:AcquireRole` API with the template's ARN and pass the parameters that the template requires.

A role template's definitions can change between versions. To see the current trust policy and permissions for a template, retrieve it with `iam:GetRoleTemplateVersion` using the template ARN. AWS recommends inspecting the template before relying on it, especially for a template that grants broad access.

How role manager uses templates

When creating a resource in a supported service console, role manager provides a role from the template that matches the service and use case. Role manager chooses the template and supplies the parameters for you. The result is the same role that you would get calling `iam:AcquireRole`. To enable role manager, see Manage access to role manager.

Role template directory

The following table lists the role template for each service that role manager supports, with the template ARN and the parameters it takes. Use the ARN to create a role from the template, or to review what role manager grants for that service. Services are listed alphabetically.

Service	Role template	Template ARN	Parameters
AWS Backup	AWSBackupDefaultServiceRoleTemplate	`arn:aws:iam::aws:role-template/backup.amazonaws.com/AWSBackupDefaultServiceRoleTemplate:1`	accountId, RoleName
Amazon Bedrock	AmazonBedrockCloudwatchIntegrationRoleTemplate	`arn:aws:iam::aws:role-template/bedrock.amazonaws.com/AmazonBedrockCloudwatchIntegrationRoleTemplate:1`	RoleName, accountId, logGroup, region
Amazon Bedrock	AmazonBedrockImportModelRoleTemplate	`arn:aws:iam::aws:role-template/bedrock.amazonaws.com/AmazonBedrockImportModelRoleTemplate:2`	CREATE_NETWORK_INTERFACE_ENABLED, RoleName, accountId, bucket_name_and_prefix, region, resourceAccountId, security_

Service	Role template	Template ARN	Parameters
			group_ids, subnet_ids
Amazon Bedrock	AmazonBedrockModelEvaluationRoleTemplate	arn:aws:iam::aws:role-template/bedrock.amazonaws.com/AmazonBedrockModelEvaluationRoleTemplate:1	RoleName, accountId, region
AWS CloudFormation	PowerUserRoleTemplate	arn:aws:iam::aws:role-template/iam.amazonaws.com/PowerUserRoleTemplate:1	AWSServiceName, RoleName
Amazon Cognito	AmazonCognitoIdentityPoolRoleTemplate	arn:aws:iam::aws:role-template/cognito-idp.amazonaws.com/AmazonCognitoIdentityPoolRoleTemplate:1	AUTHENTICATED_FLOW, identityPoolId, RoleName
Amazon Cognito	AmazonCognitoIdentityPoolPushSynchronizationRoleTemplate	arn:aws:iam::aws:role-template/cognito-sync.amazonaws.com/AmazonCognitoIdentityPoolPushSynchronizationRoleTemplate:1	RoleName, accountId, platformApplications, region
Amazon Cognito	AmazonCognitoIdentityPoolStreamsRoleTemplate	arn:aws:iam::aws:role-template/cognito-sync.amazonaws.com/AmazonCognitoIdentityPoolStreamsRoleTemplate:1	RoleName, accountId, streamName, streamRegion

Service	Role template	Template ARN	Parameters
Amazon Cognito	AmazonCognitoUserPoolSmsConfiguratioRoleTemplate	arn:aws:iam::aws:role-template/cognito-idp.amazonaws.com/AmazonCognitoUserPoolSmsConfigurationRoleTemplate:1	RoleName, externalId
Amazon Cognito	AmazonCognitoUserPoolUserImportRoleTemplate	arn:aws:iam::aws:role-template/cognito-idp.amazonaws.com/AmazonCognitoUserPoolUserImportRoleTemplate:1	accountid, region, RoleName
Amazon EKS	AmazonEKSAutoClusterRoleTemplate	arn:aws:iam::aws:role-template/eks.amazonaws.com/AmazonEKSAutoClusterRoleTemplate:1	RoleName
Amazon EKS	AmazonEKSAutoNodeRoleTemplate	arn:aws:iam::aws:role-template/ec2.amazonaws.com/AmazonEKSAutoNodeRoleTemplate:1	RoleName
AWS Elastic Beanstalk	PowerUserRoleTemplate	arn:aws:iam::aws:role-template/iam.amazonaws.com/PowerUserRoleTemplate:1	AWSServiceName, RoleName
Amazon EventBridge	PowerUserRoleTemplate	arn:aws:iam::aws:role-template/iam.amazonaws.com/PowerUserRoleTemplate:1	AWSServiceName, RoleName
AWS Glue	AWSGlueCrawlerS3PathsRoleTemplate	arn:aws:iam::aws:role-template/glue.amazonaws.com/AWSGlueCrawlerS3PathsRoleTemplate:1	RoleName, accountId, s3path

Service	Role template	Template ARN	Parameters
AWS Glue	AWSGluePowerUserRoleTemplate	arn:aws:iam::aws:role-template/glue.amazonaws.com/AWSGluePowerUserRoleTemplate:1	accountId, RoleName
AWS Lambda	PowerUserRoleTemplate	arn:aws:iam::aws:role-template/iam.amazonaws.com/PowerUserRoleTemplate:1	AWSServiceName, RoleName
Amazon RDS	AmazonRDSDirectoryServiceIntegrationRoleTemplate	arn:aws:iam::aws:role-template/rds.amazonaws.com/AmazonRDSDirectoryServiceIntegrationRoleTemplate:1	RoleName, accountId
Amazon RDS	AmazonRDSExportToS3RoleTemplate	arn:aws:iam::aws:role-template/rds.amazonaws.com/AmazonRDSExportToS3RoleTemplate:1	RoleName, accountId, bucketName
Amazon RDS	AmazonRDSImportFromS3RoleTemplate	arn:aws:iam::aws:role-template/rds.amazonaws.com/AmazonRDSImportFromS3RoleTemplate:1	CUSTOM_KMS_KEY, RoleName, accountId, bucketName, keyAccountId, keyRegion, kmsKeyId, prefix
Amazon RDS	AmazonRDSMonitoringRoleTemplate	arn:aws:iam::aws:role-template/rds.amazonaws.com/AmazonRDSMonitoringRoleTemplate:1	RoleName, accountId

Service	Role template	Template ARN	Parameters
Amazon RDS	AmazonRDSProxyRoleTemplate	arn:aws:iam::aws:role-template/rds.amazonaws.com/AmazonRDSProxyRoleTemplate:2	PROXY_WITH_CUSTOM_KEY, PROXY_WITH_DB_CONNECTION, PROXY_WITH_SECRET, RoleName, accountId, dbResourceId, dbUser, keyId, region, secretId
Amazon RDS	AmazonRDSSecretAccessForDMSRoleTemplate	arn:aws:iam::aws:role-template/rds.amazonaws.com/AmazonRDSSecretAccessForDMSRoleTemplate:1	CUSTOM_KMS_KEY, RoleName, accountId, kmsKeyId, region, secretId
Amazon SageMaker Unified Studio	AmazonSageMakerAdminIAMPermissiveExecutionRoleTemplate	arn:aws:iam::aws:role-template/datazone.amazonaws.com/AmazonSageMakerAdminIAMPermissiveExecutionRoleTemplate:1	CMK_ENABLED, RoleName, accountId, keyAccountId, keyRegion, kmsKeyId
Amazon SageMaker Unified Studio	AmazonSageMakerUserIAMPermissiveExecutionRoleTemplate	arn:aws:iam::aws:role-template/datazone.amazonaws.com/AmazonSageMakerUserIAMPermissiveExecutionRoleTemplate:1	RoleName, accountId

Service	Role template	Template ARN	Parameters
AWS Secrets Manager	AWSSecretsManagerRotationRoleTemplate	`arn:aws:iam::aws:role-template/secretsmanager.amazonaws.com/AWSSecretsManagerRotationRoleTemplate:1`	ADMIN_RESOURCE_ENABLED, CMK_ENABLED, RoleName, accountId, adminType, kmsKeyArn, region, resourceType
Amazon CloudWatch	PowerUserRoleTemplate	`arn:aws:iam::aws:role-template/iam.amazonaws.com/PowerUserRoleTemplate:1`	AWSServiceName, RoleName

Related information

- Create roles automatically with role manager
- Manage access to role manager
- IAM API Reference

IAM role management

Before a user, application, or service can use a role that you created, you must grant permissions to switch to the role. You can use any policy attached to groups or users to grant the necessary permissions. This section describes how to grant users permission to use a role. It also explains how the user can switch to a role from the AWS Management Console, the Tools for Windows PowerShell, the AWS Command Line Interface (AWS CLI) and the `AssumeRole` API.

> **Important**
>
> When you create a role programmatically instead of in the IAM console, you have an option to add a `Path` of up to 512 characters in addition to the `RoleName`, which can be up to 64 characters long. However, if you intend to use a role with the **Switch Role** feature in

the AWS Management Console, then the combined `Path` and `RoleName` cannot exceed 64 characters.

Topics

- View role access
- Generate a policy based on access information
- Grant a user permissions to switch roles
- Grant a user permissions to pass a role to an AWS service
- Revoke IAM role temporary security credentials
- Update a service-linked role
- Update a role trust policy
- Update permissions for a role
- Update settings for a role
- Delete roles or instance profiles

View role access

Before you change the permissions for a role, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Generate a policy based on access information

You might sometimes grant permissions to an IAM entity (user or role) beyond what they require. To help you refine the permissions that you grant, you can generate an IAM policy that is based on the access activity for an entity. IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that have been used by the entity in your specified date range. You can use the template to create a managed policy with fine-grained permissions and then attach it to the IAM entity. That way, you grant only the permissions that the user or role needs to interact with AWS resources for your specific use case. To learn more, see IAM Access Analyzer policy generation.

Grant a user permissions to switch roles

When an administrator creates a role for cross-account access, they establish trust between the account that owns the role, the resources (trusting account), and the account that contains the users (trusted account). To do this, the administrator of the trusting account specifies the trusted account number as the `Principal` in the role's trust policy. That *potentially* allows any user in the trusted account to assume the role. To complete the configuration, the administrator of the trusted account must give specific groups or users in that account permission to switch to the role.

To grant permission to switch to a role

1. As the administrator of the trusted account, create a new policy for the user, or edit an existing policy to add the required elements. For details, see Creating or editing the policy.
2. Then, choose how you want to share the role information:

   - **Role link:** Send users a link that takes them to the **Switch Role** page with all the details already filled in.
   - **Account ID or alias:** Provide each user with the role name along with the account ID number or account alias. The user then goes to the **Switch Role** page and adds the details manually.

   For details, see Providing information to the user.

Note that you can switch roles only when you sign in as an IAM user, a SAML-federated role, or a web-identity federated role. You cannot switch roles when you sign in as the AWS account root user.

Important

You cannot switch roles in the AWS Management Console to a role that requires an ExternalId value. You can switch to such a role only by calling the `AssumeRole` API that supports the `ExternalId` parameter.

Notes

- This topic discusses policies for a *user*, because you are ultimately granting permissions to a user to accomplish a task. However, we don't recommend that you grant permissions

directly to an individual user. When a user assumes a role, they are assigned the permissions associated with that role.

- When you switch roles in the AWS Management Console, the console always uses your original credentials to authorize the switch. This applies whether you sign in as an IAM user, as a SAML-federated role, or as a web-identity federated role. For example, if you switch to RoleA, IAM uses your original user or federated role credentials to determine if you are allowed to assume RoleA. If you then try to switch to RoleB *while you are using RoleA*, your **original** user or federated role credentials are used to authorize your attempt. The credentials for RoleA are not used for this action.

Topics

- Creating or editing the policy
- Providing information to the user

Creating or editing the policy

A policy that grants a user permission to assume a role must include a statement with the `Allow` effect on the following:

- The `sts:AssumeRole` action
- The Amazon Resource Name (ARN) of the role in a `Resource` element

Users that get the policy are allowed to switch roles on the resource listed (either through group membership or directly attached).

Note

If `Resource` is set to `*`, the user can assume any role in any account that trusts the user's account. (In other words, the role's trust policy specifies the user's account as `Principal`). As a best practice, we recommend that you follow the principle of least privilege and specify the complete ARN for only the roles that the user needs.

The following example shows a policy that lets the user assume roles in only one account. In addition, the policy uses a wildcard (*) to specify that the user can switch to a role only if the role name begins with the letters `Test`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::111122223333:role/Test*"
    }
}
```

Note

The permissions that the role grants to the user do not add to the permissions already granted to the user. When a user switches to a role, the user temporarily gives up his or her original permissions in exchange for those granted by the role. When the user exits the role, then the original user permissions are automatically restored. For example, let's say the user's permissions allow working with Amazon EC2 instances, but the role's permissions policy does not grant those permissions. In that case, while using the role, the user cannot work with Amazon EC2 instances in the console. In addition, temporary credentials obtained via `AssumeRole` do not work with Amazon EC2 instances programmatically.

Providing information to the user

After you create a role and grant your user permissions to switch to it, you must provide the user with the following:

- The name of the role
- The ID or alias of the account that contains the role

You can streamline access for your users by sending them a link that is preconfigured with the account ID and role name. You can see the role link after completing the **Create Role** wizard by selecting the **View Role** banner, or on the **Role Summary** page for any cross-account enabled role.

You can also use the following format to manually construct the link. Substitute your account ID or alias and the role name for the two parameters in the following example.

`https://signin.aws.amazon.com/switchrole?account=`*`your_account_ID_or_alias`*`&roleName=`*`optional_path/role_name`*

We recommend that you direct your users to Switch from a user to an IAM role (console) to walk them through the process. To troubleshoot common issues that you might encounter when you assume a role, see I can't assume a role.

Considerations

- If you create the role programmatically, you can create the role with a path and a name. If you do so, you must provide the complete path and role name to your users so they can enter it on the **Switch Role** page of the AWS Management Console. For example: `division_abc/subdivision_efg/role_XYZ`.
- If you create the role programmatically, you can add a `Path` of up to 512 characters and a `RoleName`. The role name can be up to 64 characters long. However, to use a role with the **Switch Role** feature in the AWS Management Console, the combined `Path` and `RoleName` cannot exceed 64 characters.

- For security purposes, you can review AWS CloudTrail logs to learn who performed an action in AWS. You can use the `sts:SourceIdentity` condition key in the role trust policy to require users to specify an identity when they assume a role. For example, you can require that IAM users specify their own user name as their source identity. This can help you determine which user performed a specific action in AWS. For more information, see `sts:SourceIdentity`. You can also use `sts:RoleSessionName` to require users to specify a session name when they assume a role. This can help you differentiate between role sessions when a role is used by different principals.

Grant a user permissions to pass a role to an AWS service

To configure many AWS services, you must *pass* an IAM role to the service. This allows the service to assume the role later and perform actions on your behalf. For most services, you only have to

pass the role to the service once during setup, and not every time that the service assumes the role. For example, assume that you have an application running on an Amazon EC2 instance. That application requires temporary credentials for authentication, and permissions to authorize the application to perform actions in AWS. When you set up the application, you must pass a role to Amazon EC2 to use with the instance that provides those credentials. You define the permissions for the applications running on the instance by attaching an IAM policy to the role. The application assumes the role every time it needs to perform the actions that are allowed by the role.

To pass a role (and its permissions) to an AWS service, a user must have permissions to *pass the role* to the service. This helps administrators ensure that only approved users can configure a service with a role that grants permissions. To allow a user to pass a role to an AWS service, you must grant the `PassRole` permission to the user's IAM user, role, or group.

Warning

- You can only use the `PassRole` permission to pass an IAM role to a service that shares the same AWS account. To pass a role in Account A to a service in Account B, you must first create an IAM role in Account B that can assume the role from Account A, and then the role in Account B can be passed to the service. For details, see Cross account resource access in IAM.
- Do not try to control who can pass a role by tagging the role and then using the `ResourceTag` condition key in a policy with the `iam:PassRole` action. This approach does not have reliable results.

When setting the `PassRole` permission, you should make sure that a user doesn't pass a role where the role has more permissions than you want the user to have. For example, Alice might not be allowed to perform any Amazon S3 actions. If Alice could pass a role to a service that allows Amazon S3 actions, the service could perform Amazon S3 actions on behalf of Alice when executing the job.

When you specify a service-linked role, you must also have permission to pass that role to the service. Some services automatically create a service-linked role in your account when you perform an action in that service. For example, Amazon EC2 Auto Scaling creates the `AWSServiceRoleForAutoScaling` service-linked role for you when you create an Auto Scaling group for the first time. If you try to specify the service-linked role when you create an Auto Scaling group and you don't have the `iam:PassRole` permission, you receive an error. If you

don't explicitly specify the role, the `iam:PassRole` permission is not required, and the default is to use `AWSServiceRoleForAutoScaling` role for all operations that are performed on that group. To learn which services support service-linked roles, see AWS services that work with IAM. To learn which services automatically create a service-linked role when you perform an action in that service, choose the **Yes** link and view the service-linked role documentation for the service.

A user can pass a role ARN as a parameter in any API operation that uses the role to assign permissions to the service. The service then checks whether that user has the `iam:PassRole` permission. To limit the user to passing only approved roles, you can filter the `iam:PassRole` permission with the `Resources` element of the IAM policy statement.

You can use the `Condition` element in a JSON policy to test the value of keys included in the request context of all AWS requests. To learn more about using condition keys in a policy, see IAM JSON policy elements: Condition. The `iam:PassedToService` condition key can be used to specify the service principal of the service to which a role can be passed. To learn more about using the `iam:PassedToService` condition key in a policy, see iam:PassedToService.

Example 1

Suppose you want to grant a user the ability to pass any of an approved set of roles to the Amazon EC2 service upon launching an instance. You need three elements:

- An IAM *permissions policy* attached to the role that determines what the role can do. Scope permissions to only the actions that the role must perform, and to only the resources that the role needs for those actions. You can use an AWS managed or customer-created IAM permissions policy.

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": {
          "Effect": "Allow",
          "Action": [ "A list of the permissions the role is allowed to use" ],
          "Resource": [ "A list of the resources the role is allowed to access" ]
      }
  }
  ```

- A *trust policy* for the role that allows the service to assume the role. For example, you could attach the following trust policy to the role with the `UpdateAssumeRolePolicy` action. This trust policy allows Amazon EC2 to use the role and the permissions attached to the role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid":
 "TrustPolicyStatementThatAllowsEC2ServiceToAssumeTheAttachedRole",
        "Effect": "Allow",
        "Principal": { "Service": "ec2.amazonaws.com" },
       "Action": "sts:AssumeRole"
    }
}
```

- An IAM *permissions policy* attached to the IAM user that allows the user to pass only those approved roles. You usually add `iam:GetRole` to `iam:PassRole` so the user can get the details of the role to be passed. In this example, the user can pass only roles that exist in the specified account with names beginning with `EC2-roles-for-XYZ-`:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/EC2-roles-for-XYZ-*"
        }
    ]
}
```

Now the user can start an Amazon EC2 instance with an assigned role. Applications running on the instance can access temporary credentials for the role through the instance profile metadata. The permissions policies attached to the role determine what the instance can do.

Example 2

Amazon Relational Database Service (Amazon RDS) supports a feature called **Enhanced Monitoring**. This feature enables Amazon RDS to monitor a database instance using an agent. It also allows Amazon RDS to log metrics to Amazon CloudWatch Logs. To enable this feature, you must create a service role to give Amazon RDS permissions to monitor and write metrics to your logs.

To create a role for Amazon RDS enhanced monitoring

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, and then choose **Create role**.
3. Choose the **AWS Service** role type, and then for **Use cases for other AWS services**, choose the **RDS** service. Choose **RDS – Enhanced Monitoring**, and then choose **Next**.
4. Choose the **AmazonRDSEnhancedMonitoringRole** permissions policy.
5. Choose **Next**.
6. For **Role name**, enter a role name that helps you identify the purpose of this role. Role names must be unique within your AWS account. When a role name is used in a policy or as part of an ARN, the role name is case sensitive. When a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive. Because various entities might reference the role, you can't edit the name of the role after it is created.
7. (Optional) For **Description**, enter a description for the new role.
8. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
9. Review the role and then choose **Create role**.

The role automatically gets a trust policy that grants the `monitoring.rds.amazonaws.com` service permissions to assume the role. After it does, Amazon RDS can perform all of the actions that the `AmazonRDSEnhancedMonitoringRole` policy allows.

The user that you want to access Enhanced Monitoring needs a policy that includes a statement that allows the user to list the RDS roles and a statement that allows the user to pass the role, like the following. Use your account number and replace the role name with the name you provided in step 6.

```
    {
```

```
      "Sid": "PolicyStatementToAllowUserToListRoles",
      "Effect": "Allow",
      "Action": ["iam:ListRoles"],
      "Resource": "*"
    },
    {
        "Sid": "PolicyStatementToAllowUserToPassOneSpecificRole",
        "Effect": "Allow",
        "Action": [ "iam:PassRole" ],
        "Resource": "arn:aws:iam::account-id:role/RDS-Monitoring-Role"
    }
```

You can combine this statement with statements in another policy or put it in its own policy. To instead specify that the user can pass any role that begins with RDS-, you can replace the role name in the resource ARN with a wildcard, as follows.

```
        "Resource": "arn:aws:iam::account-id:role/RDS-*"
```

`iam:PassRole` actions in AWS CloudTrail logs

`PassRole` is not an API call. `PassRole` is a permission, meaning no CloudTrail logs are generated for IAM `PassRole`. To review what roles are passed to which AWS services in CloudTrail, you must review the CloudTrail log that created or modified the AWS resource receiving the role. For example, a role is passed to an AWS Lambda function when it's created. The log for the `CreateFunction` action shows a record of role that was passed to the function.

Revoke IAM role temporary security credentials

> **Warning**
>
> If you follow the steps on this page, all users with current sessions created by assuming the role are denied access to all AWS actions and resources. This can result in users losing unsaved work.

When you permit users to access the AWS Management Console with a long session duration time (such as 12 hours), their temporary credentials do not expire as quickly. If users inadvertently expose their credentials to an unauthorized third-party, that party has access for the duration of the session. However, you can immediately revoke all permissions to the role's credentials

issued before a certain point in time if you need to. All temporary credentials for that role issued before the specified time become invalid. This forces all users to re-authenticate and request new credentials.

> **Note**
>
> You cannot revoke the session for a *service-linked role*.

When you revoke permissions for a role using the procedure in this topic, AWS attaches a new inline policy to the role that denies all permissions to all actions. It includes a condition that applies the restrictions only if the user assumed the role *before* the point in time when you revoke the permissions. If the user assumes the role *after* you revoked the permissions, then the deny policy does not apply to that user.

For more information on denying access, see Disabling permissions for temporary security credentials.

> **Important**
>
> This deny policy applies to all users of the specified role, not just those with longer duration console sessions.

Minimum permissions to revoke session permissions from a role

To successfully revoke session permissions from a role, you must have the `PutRolePolicy` permission for the role. This allows you to attach the `AWSRevokeOlderSessions` inline policy to the role.

Revoking session permissions

You can revoke the session permissions from a role to deny all permissions to any user who assumed the role.

> **Note**
>
> You can't edit roles in IAM that were created from IAM Identity Center permission sets. You must revoke the active permission set session for a user in IAM Identity Center. For more

information, see Revoke active IAM role sessions created by permission sets in the *IAM Identity Center User Guide*.

To immediately deny all permissions to any current user of role credentials

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then choose the name (not the checkbox) of the role whose permissions you want to revoke.
3. On the **Summary** page for the selected role, choose the **Revoke sessions** tab.
4. On the **Revoke sessions** tab, choose **Revoke active sessions**.
5. AWS asks you to confirm the action. Select the **I acknowledge that I am revoking all active sessions for this role.** checkbox and choose **Revoke active sessions** on the dialog box.

   IAM then attaches a policy named `AWSRevokeOlderSessions` to the role. After you choose **Revoke active sessions**, the policy denies all access to users who assumed the role in the past as well as approximately 30 seconds into the future. This future time choice takes into account the propagation delay of the policy in order to deal with a new session that was acquired or renewed before the updated policy is in effect in a given region. Any user who assumes the role more than approximately 30 seconds after you choose Revoke active sessions is not affected. To learn why changes are not always immediately visible, see Changes that I make are not always immediately visible.

Note

If you choose to **Revoke active sessions** again later, the date and time stamp in the policy is refreshed and it again denies all permissions to any user who assumed the role before the new specified time.

Valid users whose sessions are revoked in this way must acquire temporary credentials for a new session to continue working. The AWS CLI caches credentials until they expire. To force the CLI to delete and refresh cached credentials that are no longer valid, run one of the following commands:

Linux, macOS, or Unix

```
$ rm -r ~/.aws/cli/cache
```

Windows

```
C:\> del /s /q %UserProfile%\.aws\cli\cache
```

Revoking session permissions before a specified time

You can also revoke session permissions at any time of your choice using the AWS CLI or SDK to specify a value for the `aws:TokenIssueTime` key in the Condition element of a policy.

This policy denies all permissions when the value of `aws:TokenIssueTime` is earlier than the specified date and time. The value of `aws:TokenIssueTime` corresponds to the exact time at which the temporary security credentials were created. The `aws:TokenIssueTime` value is only present in the context of AWS requests that are signed with temporary security credentials, so the Deny statement in the policy does not affect requests that are signed with the long-term credentials of the IAM user.

This policy can also be attached to a role. In that case, the policy affects only the temporary security credentials that were created by the role before the specified date and time.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
      "DateLessThan": {"aws:TokenIssueTime": "2014-05-07T23:47:00Z"}
    }
  }
}
```

Valid users whose sessions are revoked in this way must acquire temporary credentials for a new session to continue working. The AWS CLI caches credentials until they expire. To force the CLI to delete and refresh cached credentials that are no longer valid, run one of the following commands:

Linux, macOS, or Unix

```
$ rm -r ~/.aws/cli/cache
```

Windows

```
C:\> del /s /q %UserProfile%\.aws\cli\cache
```

Update a service-linked role

The method that you use to edit a service-linked role depends on the service. Some services might allow you to edit the permissions for a service-linked role from the service console, API, or CLI. However, after you create a service-linked role, you cannot change the name of the role because various entities might reference the role. You can edit the description of any role from the IAM console, API, or CLI.

For information about which services support using service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. To learn whether the service supports editing the service-linked role, choose the **Yes** link to view the service-linked role documentation for that service.

Editing a service-linked role description (console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Enter a new description in the box and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS CLI to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, run the following commands:

```
aws iam get-role --role-name ROLE-NAME
```

Use the role name, not the ARN, to refer to roles with the CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.

2. To update a service-linked role's description, run the following command:

```
aws iam update-role --role-name ROLE-NAME --description OPTIONAL-DESCRIPTION
```

Editing a service-linked role description (AWS API)

You can use the AWS API to edit the description of a service-linked role.

To change the description of a service-linked role (AWS API)

1. (Optional) To view the current description for a role, call the following operation, and specify the name of the role:

   AWS API: GetRole
2. To update a role's description, call the following operation, and specify the name (and optional description) of the role:

   AWS API: UpdateRole

Update a role trust policy

To change who can assume a role, you must modify the role's trust policy. You cannot modify the trust policy for a *service-linked role*.

Notes

- If a user is listed as the principal in a role's trust policy but cannot assume the role, check the user's permissions boundary. If a permissions boundary is set for the user, then it must allow the `sts:AssumeRole` action.
- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide

compute resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically update these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again to obtain temporary credentials. However, if you intend to pass session tags or a session policy, you need to assume the current role again.

Updating a role trust policy (console)

To change a role trust policy in the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**.
3. In the list of roles in your account, choose the name of the role that you want to modify.
4. Choose the **Trust relationships** tab, and then choose **Edit trust policy**.
5. Edit the trust policy as needed. To add additional principals that can assume the role, specify them in the `Principal` element. For example, the following policy snippet shows how to reference two AWS accounts in the `Principal` element:

```
"Principal": {
  "AWS": [
    "arn:aws:iam::111122223333:root",
    "arn:aws:iam::444455556666:root"
  ]
},
```

If you specify a principal in another account, adding an account to the trust policy of a role is only half of establishing the cross-account trust relationship. By default, no users in the trusted accounts can assume the role. The administrator for the newly trusted account must grant the users the permission to assume the role. To do that, the administrator must create or edit a policy that is attached to the user to allow the user access to the `sts:AssumeRole` action. For more information, see the following procedure or Grant a user permissions to switch roles.

The following policy snippet shows how to reference two AWS services in the `Principal` element:

```
"Principal": {
  "Service": [
    "opsworks.amazonaws.com",
    "ec2.amazonaws.com"
  ]
},
```

6. When you are finished editing your trust policy, choose **Update policy** to save your changes.

   For more information about policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

To allow users in a trusted external account to use the role (console)

For more information and detail about this procedure, see Grant a user permissions to switch roles.

1. Sign in to the trusted external AWS account.
2. Decide whether to attach the permissions to a user or to a group. In the navigation pane of the IAM console, choose **Users** or **User groups** accordingly.
3. Choose the name of the user or group to which you want to grant access, and then choose the **Permissions** tab.
4. Do one of the following:

   - To edit a customer managed policy, choose the name of the policy, choose **Edit policy**, and then choose the **JSON** tab. You cannot edit an AWS managed policy. AWS managed policies appear with the AWS icon ().
     For more information about the difference between AWS managed policies and customer managed policies, see Managed policies and inline policies.
   - To edit an inline policy, choose the arrow next to the name of the policy and choose **Edit policy**.
5. In the policy editor, add a new `Statement` element that specifies the following:

```
{
```

```
  "Effect": "Allow",
  "Action": "sts:AssumeRole",
  "Resource": "arn:aws:iam::ACCOUNT-ID:role/ROLE-NAME"
}
```

Replace the ARN in the statement with the ARN of the role that the user can assume.

6. Follow the prompts on screen to finish editing the policy.

Updating a role trust policy (AWS CLI)

You can use the AWS CLI to change who can assume a role.

To modify a role trust policy (AWS CLI)

1. (Optional) If you don't know the name of the role that you want to modify, run the following command to list the roles in your account:

   - aws iam list-roles

2. (Optional) To view the current trust policy for a role, run the following command:

   - aws iam get-role

3. To modify the trusted principals that can access the role, create a text file with the updated trust policy. You can use any text editor to construct the policy.

   For example, the following trust policy shows how to reference two AWS accounts in the `Principal` element. This allows users within two separate AWS accounts to assume this role.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"AWS": [
            "arn:aws:iam::111122223333:root",
            "arn:aws:iam::444455556666:root"
        ]},
        "Action": "sts:AssumeRole"
    }
```

```
}
```

If you specify a principal in another account, adding an account to the trust policy of a role is only half of establishing the cross-account trust relationship. By default, no users in the trusted accounts can assume the role. The administrator for the newly trusted account must grant the users the permission to assume the role. To do that, the administrator must create or edit a policy that is attached to the user to allow the user access to the `sts:AssumeRole` action. For more information, see the following procedure or Grant a user permissions to switch roles.

4. To use the file that you just created to update the trust policy, run the following command:

   - aws iam update-assume-role-policy

To allow users in a trusted external account to use the role (AWS CLI)

For more information and detail about this procedure, see Grant a user permissions to switch roles.

1. Create a JSON file that contains a permissions policy that grants permissions to assume the role. For example, the following policy contains the minimum necessary permissions:

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": {
           "Effect": "Allow",
           "Action": "sts:AssumeRole",
           "Resource": "arn:aws:iam::111122223333:role/ROLE-NAME"
       }
   }
   ```

   Replace the ARN in the statement with the ARN of the role that the user can assume.

2. Run the following command to upload the JSON file that contains the trust policy to IAM:

   - aws iam create-policy

The output of this command includes the ARN of the policy. Make a note of this ARN because you will need it in a later step.

3. Decide which user or group to attach the policy to. If you don't know the name of the intended user or group, use one of the following commands to list the users or groups in your account:

   - aws iam list-users
   - aws iam list-groups

4. Use one of the following commands to attach the policy that you created in the previous step to the user or group:

   - aws iam attach-user-policy
   - aws iam attach-group-policy

Updating a role trust policy (AWS API)

You can use the AWS API to change who can assume a role.

To modify a role trust policy (AWS API)

1. (Optional) If you don't know the name of the role that you want to modify, call the following operation to list the roles in your account:

   - ListRoles

2. (Optional) To view the current trust policy for a role, call the following operation:

   - GetRole

3. To modify the trusted principals that can access the role, create a text file with the updated trust policy. You can use any text editor to construct the policy.

   For example, the following trust policy shows how to reference two AWS accounts in the `Principal` element. This allows users within two separate AWS accounts to assume this role.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
```

```
        "Effect": "Allow",
        "Principal": {"AWS": [
            "arn:aws:iam::111122223333:root",
            "arn:aws:iam::444455556666:root"
        ]},
        "Action": "sts:AssumeRole"
    }
}
```

If you specify a principal in another account, adding an account to the trust policy of a role is only half of establishing the cross-account trust relationship. By default, no users in the trusted accounts can assume the role. The administrator for the newly trusted account must grant the users the permission to assume the role. To do that, the administrator must create or edit a policy that is attached to the user to allow the user access to the `sts:AssumeRole` action. For more information, see the following procedure or Grant a user permissions to switch roles.

4. To use the file that you just created to update the trust policy, call the following operation:

   - UpdateAssumeRolePolicy

To allow users in a trusted external account to use the role (AWS API)

For more information and detail about this procedure, see Grant a user permissions to switch roles.

1. Create a JSON file that contains a permissions policy that grants permissions to assume the role. For example, the following policy contains the minimum necessary permissions:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::111122223333:role/ROLE-NAME"
    }
}
```

Replace the ARN in the statement with the ARN of the role that the user can assume.

2. Call the following operation to upload the JSON file that contains the trust policy to IAM:

   - CreatePolicy

   The output of this operation includes the ARN of the policy. Make a note of this ARN because you will need it in a later step.

3. Decide which user or group to attach the policy to. If you don't know the name of the intended user or group, call one of the following operations to list the users or groups in your account:

   - ListUsers
   - ListGroups

4. Call one of the following operations to attach the policy that you created in the previous step to the user or group:

   - API: AttachUserPolicy
   - AttachGroupPolicy

Update permissions for a role

Use the following procedures to update a role's permissions policies and permissions boundaries.

Prerequisite: View role access

Before you change the permissions for a role, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Update the permissions policy for a role

To change the permissions allowed by the role, modify the role's permissions policy (or policies). You cannot modify the permissions policy for a *service-linked role* in IAM. You might be able to modify the permissions policy within the service that depends on the role. To check whether a service supports this feature, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Updating a role permissions policy (console)

To change the permissions allowed by a role (console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**.
3. Choose the name of the role that you want to modify, and then choose the **Permissions** tab.
4. Do one of the following:

   - To edit an existing customer managed policy, choose the name of the policy and then choose **Edit policy**.

     > **Note**
     >
     > You cannot edit an AWS managed policy. AWS managed policies appear with the AWS icon
     > ().
     > For more information about the difference between AWS managed policies and customer managed policies, see Managed policies and inline policies.

   - To attach an existing managed policy to the role, choose **Add permissions** and then choose **Attach policies**.
   - To edit an existing inline policy, expand the policy and choose **Edit**.
   - To embed a new inline policy, choose **Add permissions** and then choose **Create inline policy**.
   - To remove an existing policy from the role, select the check box next to the policy name and then choose **Remove**.

Updating a role permissions policy (AWS CLI)

To change the permissions allowed by the role, modify the role's permissions policy (or policies). You cannot modify the permissions policy for a *service-linked role* in IAM. You might be able to modify the permissions policy within the service that depends on the role. To check whether a service supports this feature, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

To change the permissions allowed by a role (AWS CLI)

1. (Optional) To view the current permissions associated with a role, run the following commands:

   1. aws iam list-role-policies to list inline policies
   2. aws iam list-attached-role-policies to list managed policies

2. The command to update permissions for the role differs depending on whether you are updating a managed policy or an inline policy.

   To update a managed policy, run the following command to create a new version of the managed policy:

   - aws iam create-policy-version

   To update an inline policy, run the following command:

   - aws iam put-role-policy

Updating a role permissions policy (AWS API)

To change the permissions allowed by the role, modify the role's permissions policy (or policies). You cannot modify the permissions policy for a *service-linked role* in IAM. You might be able to modify the permissions policy within the service that depends on the role. To check whether a service supports this feature, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

To change the permissions allowed by a role (AWS API)

1. (Optional) To view the current permissions associated with a role, call the following operations:

   1. ListRolePolicies to list inline policies
   2. ListAttachedRolePolicies to list managed policies

2. The operation to update permissions for the role differs depending on whether you are updating a managed policy or an inline policy.

To update a managed policy, call the following operation to create a new version of the managed policy:

- CreatePolicyVersion

To update an inline policy, call the following operation:

- PutRolePolicy

Update the permissions boundary for a role

To change the maximum permissions allowed for a role, modify the role's permissions boundary.

Updating a role permissions boundary (console)

To change the policy used to set the permissions boundary for a role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose the name of the role with the permissions boundary that you want to change.
4. Choose the **Permissions** tab. If necessary, open the **Permissions boundary** section and then choose **Change boundary**.
5. Select the policy that you want to use for the permissions boundary.
6. Choose **Change boundary**.

   Your changes don't take effect until the next time someone assumes this role.

Updating a role permissions boundary (AWS CLI)

To change the managed policy used to set the permissions boundary for a role (AWS CLI)

1. (Optional) To view the current permissions boundary for a role, run the following command:

   - aws iam get-role
2. To use a different managed policy to update the permissions boundary for a role, run the following command:

- aws iam put-role-permissions-boundary

A role can have only one managed policy set as a permissions boundary. If you change the permissions boundary, you change the maximum permissions allowed for a role.

Updating a role permissions boundary (AWS API)

To change the managed policy used to set the permissions boundary for a role (AWS API)

1. (Optional) To view the current permissions boundary for a role, call the following operation:
   - GetRole
2. To use a different managed policy to update the permissions boundary for a role, call the following operation:
   - PutRolePermissionsBoundary

   A role can have only one managed policy set as a permissions boundary. If you change the permissions boundary, you change the maximum permissions allowed for a role.

Update settings for a role

Use the following procedures to update a role's description or change the maximum session duration for a role.

Update a role description

To change the description of the role, modify the description text.

Updating a role description (console)

To change the description of a role (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**.
3. Choose the name of the role to modify.

4. In the **Summary** section, choose **Edit**.
5. Enter a new description in the box and choose **Save changes**.

Updating a role description (AWS CLI)

To change the description of a role (AWS CLI)

1. (Optional) To view the current description for a role, run the following command:
   - aws iam get-role
2. To update a role's description, run the following command with the description parameter:
   - aws iam update-role

Updating a role description (AWS API)

To change the description of a role (AWS API)

1. (Optional) To view the current description for a role, call the following operation:
   - GetRole
2. To update a role's description, call the following operation with the description parameter:
   - UpdateRole

Update the maximum session duration for a role

To specify the maximum session duration setting for roles that are assumed using the console, the AWS CLI, or AWS API, modify the maximum session duration setting value. This setting can have a value from 1 hour to 12 hours. If you do not specify a value, the default maximum of 1 hour is applied. This setting does not limit sessions assumed by AWS services.

Updating the maximum role session duration (console)

To change the maximum session duration setting for roles that are assumed using the console, AWS CLI, or AWS API (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane of the IAM console, choose **Roles**.
3. Choose the name of the role to modify.
4. In the **Summary** section, choose **Edit**.
5. For **Maximum session duration**, choose a value. Alternatively, choose **Custom duration** and enter a value (in seconds).
6. Choose **Save changes**.

   Your changes don't take effect until the next time someone assumes this role. To learn how to revoke existing sessions for this role, see Revoke IAM role temporary security credentials.

In the AWS Management Console, IAM user sessions are 12 hours by default. IAM users who switch roles in the console are granted the role maximum session duration, or the remaining time in the user's session, whichever is less.

Anyone who assumes the role from the AWS CLI or AWS API can request a longer session, up to this maximum. The `MaxSessionDuration` setting determines the maximum duration of the role session that can be requested.

- To specify a session duration using the AWS CLI use the `duration-seconds` parameter. To learn more, see Switch to an IAM role (AWS CLI).
- To specify a session duration using the AWS API, use the `DurationSeconds` parameter. To learn more, see Switch to an IAM role (AWS API).

Updating the maximum role session duration (AWS CLI)

> **Note**
>
> Anyone who assumes the role from the AWS CLI or API can use the `duration-seconds` CLI parameter or the `DurationSeconds` API parameter to request a longer session. The `MaxSessionDuration` setting determines the maximum duration of the role session that can be requested using the `DurationSeconds` parameter. If users don't specify a value for the `DurationSeconds` parameter, their security credentials are valid for one hour.

To change the maximum session duration setting for roles that are assumed using the AWS CLI (AWS CLI)

1. (Optional) To view the current maximum session duration setting for a role, run the following command:

   - aws iam get-role

2. To update a role's maximum session duration setting, run the following command with the `max-session-duration` CLI parameter or the `MaxSessionDuration` API parameter:

   - aws iam update-role

   Your changes don't take effect until the next time someone assumes this role. To learn how to revoke existing sessions for this role, see Revoke IAM role temporary security credentials.

Updating the maximum role session duration (AWS API)

> **Note**
>
> Anyone who assumes the role from the AWS CLI or API can use the `duration-seconds` CLI parameter or the `DurationSeconds` API parameter to request a longer session. The `MaxSessionDuration` setting determines the maximum duration of the role session that can be requested using the `DurationSeconds` parameter. If users don't specify a value for the `DurationSeconds` parameter, their security credentials are valid for one hour.

To change the maximum session duration setting for roles that are assumed using the API (AWS API)

1. (Optional) To view the current maximum session duration setting for a role, call the following operation:

   - GetRole

2. To update a role's maximum session duration setting, call the following operation with the `max-session-duration` CLI parameter or the `MaxSessionDuration` API parameter:

   - UpdateRole

Your changes don't take effect until the next time someone assumes this role. To learn how to revoke existing sessions for this role, see Revoke IAM role temporary security credentials.

Delete roles or instance profiles

If you no longer need a role, we recommend that you delete the role and its associated permissions. That way you don't have an unused entity that is not actively monitored or maintained.

If the role was associated with an EC2 instance, you can also remove the role from the instance profile and then delete the instance profile.

> **Warning**
>
> Make sure that you do not have any Amazon EC2 instances running with the role or instance profile you are about to delete. Deleting a role or instance profile that is associated with a running instance will break any applications that are running on the instance.

If you prefer not to permanently delete a role, you can disable a role. To do this, change the role policies and then revoke all current sessions. For example, you could add a policy to the role that denied access to all of AWS. You could also edit the trust policy to deny access to anyone attempting to assume the role. For more information about revoking sessions, see Revoke IAM role temporary security credentials.

Topics

- Viewing role access
- Deleting a service-linked role
- Deleting an IAM role (console)
- Deleting an IAM role (AWS CLI)
- Deleting an IAM role (AWS API)
- Related information

Viewing role access

Before you delete a role, we recommend that you review when the role was last used. You can do this using the AWS Management Console, the AWS CLI, or the AWS API. You should view this information because you don't want to remove access from someone using the role.

The date of the role last activity might not match the last date reported in the **Last Accessed** tab. The **Last Accessed** tab reports activity only for services allowed by the role permissions policies. The date of the role last activity includes the last attempt to access any service in AWS.

> **Note**
>
> The tracking period for a role last activity and Last Accessed data is for the trailing 400 days. This period can be shorter if your Region began supporting these features within the last year. The role might have been used more than 400 days ago. For more information about the tracking period, see Where AWS tracks last accessed information.

To view when a role was last used (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Find the row of the role with the activity you want to view. You can use the search field to narrow the results. View the **Last activity** column to see the number of days since the role was last used. If the role has not been used within the tracking period, then the table displays **None**.
4. Choose the name of the role to view more information. The role **Summary** page also includes **Last activity**, which displays the last used date for the role. If the role has not been used within the last 400 days, then **Last activity** displays **Not accessed in the tracking period**.

To view when a role was last used (AWS CLI)

`aws iam get-role` - Run this command to return information about a role, including the `RoleLastUsed` object. This object contains the `LastUsedDate` and the `Region` in which the role was last used. If `RoleLastUsed` is present but does not contain a value, then the role has not been used within the tracking period.

To view when a role was last used (AWS API)

`GetRole` - Call this operation to return information about a role, including the `RoleLastUsed` object. This object contains the `LastUsedDate` and the `Region` in which the role was last used. If `RoleLastUsed` is present but does not contain a value, then the role has not been used within the tracking period.

Deleting a service-linked role

The method that you use to delete a service-linked role depends on the service. In some cases, you don't need to manually delete a service-linked role. For example, when you complete a specific action (such as removing a resource) in the service, the service might delete the service-linked role for you. In other cases, the service might support deleting a service-linked role manually from the service console, API, or AWS CLI.

Review the documentation for the *service-linked role* in the linked service to learn how to delete the role. You can view the service-linked roles in your account by going to the IAM **Roles** page in the console. Service-linked roles appear with **(Service-linked role)** in the **Trusted entities** column of the table. A banner on the role **Summary** page also indicates that the role is a service-linked role.

If the service does not include documentation for deleting the service-linked role, you can use the IAM console, AWS CLI, or API to delete the role.

Deleting an IAM role (console)

When you use the AWS Management Console to delete a role, IAM automatically detaches managed policies associated with the role. It also automatically deletes any inline policies associated with the role, and any Amazon EC2 instance profile that contains the role.

> **Important**
>
> In some cases, a role might be associated with an Amazon EC2 instance profile, and the role and the instance profile might have the same name. In that case you can use the AWS Management Console to delete the role and the instance profile. This linkage happens automatically for roles and instance profiles that you create in the console. If you created the role from the AWS CLI, Tools for Windows PowerShell, or the AWS API, then the role and the instance profile might have different names. In that case you cannot use the console to delete them. Instead, you must use the AWS CLI, Tools for Windows

PowerShell, or AWS API to first remove the role from the instance profile. You must then take a separate step to delete the role.

To delete a role (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then select the check box next to the role name that you want to delete.
3. At the top of the page, choose **Delete**.
4. In the confirmation dialog box, review the last accessed information, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm if the role is currently active. If you want to proceed, enter the name of the role in the text input field and choose **Delete**. If you are sure, you can proceed with the deletion even if the last accessed information is still loading.

Note

You cannot use the console to delete an instance profile unless it has the same name as the role. The instance profile is deleted as part of the process of deleting a role as described in the preceding procedure. To delete an instance profile without also deleting the role, you must use the AWS CLI or AWS API. For more information, see the following sections.

Deleting an IAM role (AWS CLI)

When you use the AWS CLI to delete a role, you must first delete inline policies associated with the role. You must also detach managed policies associated with the role. If you want to delete the associated instance profile that contains the role, you must delete it separately.

To delete a role (AWS CLI)

1. If you don't know the name of the role that you want to delete, enter the following command to list the roles in your account:

```
aws iam list-roles
```

The list includes the Amazon Resource Name (ARN) of each role. Use the role name, not the ARN, to refer to roles with the CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.

2. Remove the role from all instance profiles that the role is associated with.

   a. To list all instance profiles that the role is associated with, enter the following command:

   ```
   aws iam list-instance-profiles-for-role --role-name role-name
   ```

   b. To remove the role from an instance profile, enter the following command for each instance profile:

   ```
   aws iam remove-role-from-instance-profile --instance-profile-name instance-profile-name --role-name role-name
   ```

3. Delete all policies that are associated with the role.

   a. To list all inline policies that are in the role, enter the following command:

   ```
   aws iam list-role-policies --role-name role-name
   ```

   b. To delete each inline policy from the role, enter the following command for each policy:

   ```
   aws iam delete-role-policy --role-name role-name --policy-name policy-name
   ```

   c. To list all managed policies that are attached to the role, enter the following command:

   ```
   aws iam list-attached-role-policies --role-name role-name
   ```

   d. To detach each managed policy from the role, enter the following command for each policy:

   ```
   aws iam detach-role-policy --role-name role-name --policy-arn policy-arn
   ```

4. Enter the following command to delete the role:

   ```
   aws iam delete-role --role-name role-name
   ```

5. If you do not plan to reuse the instance profiles that were associated with the role, you can enter the following command to delete them:

```
aws iam delete-instance-profile --instance-profile-name instance-profile-name
```

Deleting an IAM role (AWS API)

When you use the IAM API to delete a role, you must first delete inline policies associated with the role. You must also detach managed policies associated with the role. If you want to delete the associated instance profile that contains the role, you must delete it separately.

To delete a role (AWS API)

1. To list all instance profiles that a role is associated with, call ListInstanceProfilesForRole.

   To remove the role from an instance profile, call RemoveRoleFromInstanceProfile. You must pass the role name and instance profile name.

   If you are not going to reuse an instance profile that was associated with the role, call DeleteInstanceProfile to delete it.
2. To list all inline policies for a role, call ListRolePolicies.

   To delete inline policies that are associated with the role, call DeleteRolePolicy. You must pass the role name and inline policy name.
3. To list all managed policies that are attached to a role, call ListAttachedRolePolicies.

   To detach managed policies that are attached to the role, call DetachRolePolicy. You must pass the role name and managed policy ARN.
4. Call DeleteRole to delete the role.

Related information

For general information about instance profiles, see Use instance profiles.

For general information about service-linked roles, see Create a service-linked role.

Methods to assume a role

Before a user, application, or service can use a role that you created, you must grant permissions to switch to the role. You can use any policy attached to groups or users to grant the necessary permissions. After permissions are granted, the user can assume a role from the AWS Management

Console, the Tools for Windows PowerShell, the AWS Command Line Interface (AWS CLI) and the `AssumeRole` API.

> **Important**
>
> When you create a role programmatically instead of in the IAM console, you have an option to add a `Path` of up to 512 characters in addition to the `RoleName`, which can be up to 64 characters long. However, if you intend to use a role with the **Switch Role** feature in the AWS Management Console, then the combined `Path` and `RoleName` cannot exceed 64 characters.

The method used to assume the role determines who can assume the role and how long the role session can last. When using `AssumeRole*` API operations, the IAM role that you assume is the resource. The user or role that calls `AssumeRole*` API operations is the principal.

The following table compares methods for assuming roles.

Method of assuming the role	Who can assume the role	Method to specify credential lifetime	Credential lifetime (min | max | default)
AWS Management Console	User or roles[1](by switching roles)	**Maximum session duration** on the **Role** Summary page	15m | Maximum session duration setting[2] | 1hr
`assume-role` CLI or `AssumeRole` API operation	User or role[1]	`duration-seconds` CLI or `DurationSeconds` API parameter	15m | Maximum session duration setting[2] | 1hr
`assume-role-with-saml` CLI or `AssumeRol`	Any user authenticated using SAML	`duration-seconds` CLI or `DurationS`	15m | Maximum session duration setting[2] | 1hr

Method of assuming the role	Who can assume the role	Method to specify credential lifetime	Credential lifetime (min | max | default)
eWithSAML API operation		econds API parameter	
assume-role-with-web-identity CLI or AssumeRoleWithWebIdentity API operation	Any user authenticated using an OIDC provider	duration-seconds CLI or DurationSeconds API parameter	15m | Maximum session duration setting[2] | 1hr
Console URL constructed with AssumeRole	User or role	SessionDuration HTML parameter in the URL	15m | 12hr | 1hr
Console URL constructed with AssumeRoleWithSAML	Any user authenticated using SAML	SessionDuration HTML parameter in the URL	15m | 12hr | 1hr
Console URL constructed with AssumeRoleWithWebIdentity	Any user authenticated using an OIDC provider	SessionDuration HTML parameter in the URL	15m | 12hr | 1hr

[1] Using the credentials from one role to assume a different role is called role chaining. When you use role chaining, the role's session duration is limited to one hour. This applies to AWS Management Console role switching, AWS CLI, and API operations. This limitation does not apply

to the initial assumption of a role from user credentials, or to applications running on Amazon EC2 instances using instance profiles.

[2] This setting can have a value from 1 hour to 12 hours. For details about modifying the maximum session duration setting, see IAM role management. This setting determines the maximum session duration that you can request when you get the role credentials. For example, when you use the AssumeRole* API operations to assume a role, you can specify a session length using the `DurationSeconds` parameter. Use this parameter to specify the length of the role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role. IAM users who switch roles in the console are granted the maximum session duration, or the remaining time in their user session, whichever is less. Assume that you set a maximum duration of 5 hours on a role. An IAM user that has been signed into the console for 10 hours (out of the default maximum of 12) switches to the role. The available role session duration is 2 hours. To learn how to view the maximum value for your role, see Update the maximum session duration for a role later in this page.

Notes

- The maximum session duration setting does not limit sessions that are assumed by AWS services.
- Amazon EC2 IAM role credentials are not subject to the maximum session duration configured in the role.
- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide compute resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically update these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again to obtain temporary credentials. However, if you intend to pass session tags or a session policy, you need to assume the current role again. To learn how to modify a role trust policy to add the principal role ARN or AWS account ARN, see Update a role trust policy.

Topics

- Switch from a user to an IAM role (console)
- Switch to an IAM role (AWS CLI)

- Switch to an IAM role (Tools for Windows PowerShell)
- Switch to an IAM role (AWS API)
- Use an IAM role to grant permissions to applications running on Amazon EC2 instances
- Use instance profiles

Switch from a user to an IAM role (console)

You can switch roles when you sign in as an IAM user, a user in IAM Identity Center, a SAML-federated role, or a web-identity federated role. A *role* specifies a set of permissions that you can use to access AWS resources that you need. However, you don't sign in to a role, but once signed in as an IAM user you can switch to an IAM role. This temporarily sets aside your original user permissions and instead gives you the permissions assigned to the role. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create them, see IAM roles, and IAM role creation.

The permissions of your user and any roles that you switch to aren't cumulative. Only one set of permissions is active at a time. When you switch to a role, you temporarily give up your user permissions and work with the permissions that are assigned to the role. When you exit the role, your user permissions are automatically restored.

When you switch roles in the AWS Management Console, the console always uses your original credentials to authorize the switch. For example, if you switch to RoleA, IAM uses your original credentials to determine whether you are allowed to assume RoleA. If you then switch to RoleB *while you are using RoleA*, AWS still uses your **original** credentials to authorize the switch, not the credentials for RoleA.

> **Note**
>
> When you sign in as a user in IAM Identity Center, as a SAML-federated role, or as a web-identity federated role you assume an IAM role when you start your session. For example, when a user in IAM Identity Center signs in to the AWS access portal they can choose a permission set or access a role assigned through account access manager before they can access AWS resources.

Role sessions

When you switch roles, your AWS Management Console session lasts for 1 hour by default. IAM user sessions are 12 hours by default, other users might have different session durations defined. When you switch roles in the console, you are granted the role maximum session duration, or the remaining time in your user session, whichever is less. You can't extend your session duration by assuming a role.

For example, assume that a maximum session duration of 10 hours is set for a role. You have been signed in to the console for 8 hours when you decide to switch to the role. There are 4 hours remaining in your user session, so the allowed role session duration is 4 hours, not the maximum session duration of 10 hours. The following table shows how to determine the session duration for an IAM user when switching roles in the console.

IAM user session time remaining is...	Role session duration is...
Less than role maximum session duration	Time remaining in user session
Greater than role maximum session duration	Maximum session duration value
Equal to role maximum session duration	Maximum session duration value (approximate)

Using the credentials from one role to assume a different role is called role chaining. When you use role chaining, the session duration is limited to one hour, regardless of the maximum session duration setting configured for individual roles. This applies to AWS Management Console role switching, AWS CLI, and API operations.

> **Note**
>
> Some AWS service consoles can autorenew your role session when it expires without you taking any action. Some might prompt you to reload your browser page to reauthenticate your session.

Considerations

- You can't switch roles if you sign in as the AWS account root user.
- Users must be granted permission to switch roles by policy. For instructions, see Grant a user permissions to switch roles.
- You can't switch roles in the AWS Management Console to a role that requires an ExternalId value. You can switch to such a role only by calling the `AssumeRole` API that supports the `ExternalId` parameter.

To switch to a role

1. Follow the sign-in procedure appropriate to you user type as described in Sign in to the AWS Management Console in the *AWS Sign-In User Guide*.
2. In the AWS Management Console, choose your user name on the navigation bar in the upper right. It typically looks like this: ***username*@*account_ID_number_or_alias***.
3. Select one of the following methods to switch roles:
   - Choose **Switch role**.
   - If you have opted in to multi-session support, choose **Add session** and select **Switch role**.

   > **Note**
   >
   > You can sign in to up to five different identities simultaneously in a single web browser in the AWS Management Console. For details, see Signing in to multiple accounts in the *AWS Management Console Getting Started Guide*.
4. On the **Switch Role** page, type the account ID number or the account alias and the name of the role that was provided by your administrator.

> **Note**
>
> If your administrator created the role with a path, such as `division_abc/subdivision_efg/roleToDoX`, then you must type that complete path and name in the **Role** box. If you type only the role name, or if the combined `Path` and `RoleName` exceed 64 characters, the role switch fails. This is a limit of the browser cookies that store the role name. If this happens, contact your administrator and ask them to reduce the size of the path and role name.

5. (Optional)You can enter a display name and select a display color that will highlight the role in the console navigation bar.

   - For **Display name**, type text that you want to appear on the navigation bar in place of your user name when this role is active. A name is suggested, based on the account and role information, but you can change it to whatever has meaning for you.
   - For **Display color**, select a color to highlight the display name.

   The name and color can help remind you when this role is active, which changes your permissions. For example, for a role that gives you access to the test environment, you might specify a **Display name** of **`Test`** and select the green **Color**. For the role that gives you access to production, you might specify a **Display name** of **`Production`** and select red as the **Color**.

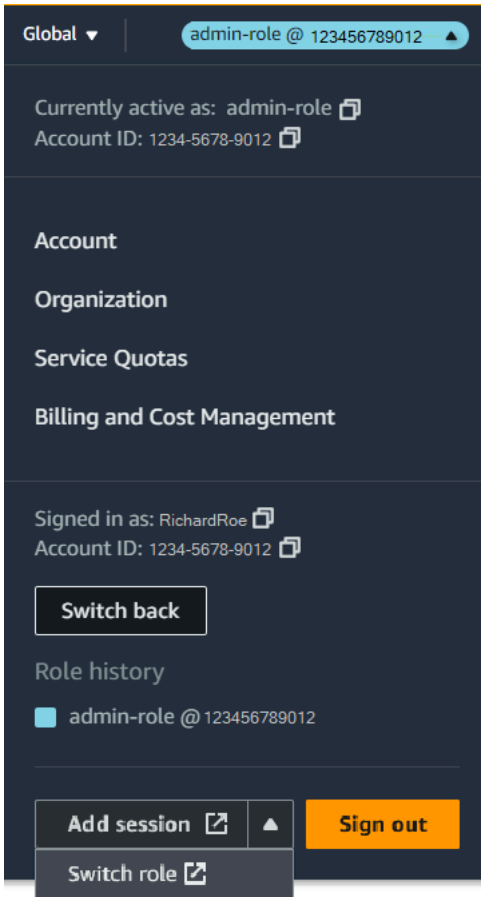
6. Choose **Switch Role**. The display name and color replace your user name on the navigation bar, and you can start using the permissions that the role grants you.

7. After you have completed the tasks that require the IAM role you can switch back to your original session. This will remove the additional permissions provided by the role and return you to your standard permissions.

   a. In the IAM console, choose your role's **Display name** on the navigation bar in the upper right.

   b. Choose **Switch back**.

      For example, assume you are signed in to account number `123456789012` using the user name `Richard`. After you use the `admin-role` role, you want to stop using the role and return to your original permissions. To stop using the role, you choose **admin-role @ 123456789012**, and then choose **Switch back**.



Tip

The last several roles that you used appear on the menu. The next time you want to switch to one of those roles, you can simply choose the role you want. You are only required to type the account and role information manually if the role isn't displayed on the menu.

Additional resources

- Grant a user permissions to switch roles
- Grant a user permissions to pass a role to an AWS service
- Create a role to give permissions to an IAM user
- Create a role to delegate permissions to an AWS service
- Troubleshoot IAM roles

Switch to an IAM role (AWS CLI)

A *role* specifies a set of permissions that you can use to access AWS resources that you need. In that sense, it is similar to a user in AWS Identity and Access Management (IAM). When you sign in as a user, you get a specific set of permissions. However, you don't sign in to a role, but after signing in as a user, you can switch to a role. This temporarily sets aside your original user permissions and instead gives you the permissions assigned to the role. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create and configure them, see IAM roles, and IAM role creation. To learn about the different methods that you can use to assume a role, see Methods to assume a role.

> **Important**
>
> The permissions of your IAM user and any roles that you assume are not cumulative. Only one set of permissions is active at a time. When you assume a role, you temporarily give up your previous user or role permissions and work with the permissions that are assigned to the role. When you exit the role, your user permissions are automatically restored.

You can use a role to run an AWS CLI command when you are signed in as an IAM user. You can also use a role to run an AWS CLI command when you are signed in as an externally authenticated user (SAML or OIDC) that is already using a role. In addition, you can use a role to run an AWS CLI command from within an Amazon EC2 instance that is attached to a role through its instance profile. You cannot assume a role when you are signed in as the AWS account root user.

Role chaining – You can also use role chaining, which is using permissions from a role to access a second role.

By default, your role session lasts for one hour. When you assume this role using the `assume-role*` CLI operations, you can specify a value for the `duration-seconds` parameter. This value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role. If you switch roles in the console, your session duration is limited to maximum of one hour. To learn how to view the maximum value for your role, see Update the maximum session duration for a role.

If you use role chaining, your session duration is limited to a maximum of one hour. If you then use the `duration-seconds` parameter to provide a value greater than one hour, the operation fails.

Example scenario: Switch to a production role

Imagine that you are an IAM user for working in the development environment. In this scenario, you occasionally need to work with the production environment at the command line with the AWS CLI. You already have an access key credential set available to you. This can be the access key pair that is assigned to your standard IAM user. Or, if you signed in as a SAML or OIDC federated principal, it can be the access key pair for the role that was initially assigned to you. If your current permissions grant you the ability to assume a specific IAM role, then you can identify that role in a "profile" in the AWS CLI configuration files. That command is then run with the permissions of the specified IAM role, not the original identity. Note that when you specify that profile in an AWS CLI command, you are using the new role. In this situation, you cannot make use of your original permissions in the development account at the same time. The reason is that only one set of permissions can be in effect at a time.

> **Note**
>
> For security purposes, administrators can review AWS CloudTrail logs to learn who performed an action in AWS. Your administrator might require that you specify a source identity or a role session name when you assume the role. For more information, see `sts:SourceIdentity` and `sts:RoleSessionName`.

To switch to a production role (AWS CLI)

1. If you have never used the AWS CLI, then you must first configure your default CLI profile. Open a command prompt and set up your AWS CLI installation to use the access key from your IAM user or from your federated role. For more information, see Configuring the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

   Run the aws configure command as follows:

   ```
   aws configure
   ```

   When prompted, provide the following information:

   ```
   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-east-2
   ```

```
Default output format [None]: json
```

2. Create a new profile for the role in the `.aws/config` file in Unix or Linux, or the `C:\Users\USERNAME\.aws\config` file in Windows. The following example creates a profile called `prodaccess` that switches to the role *`ProductionAccessRole`* in the `123456789012` account. You get the role ARN from the account administrator who created the role. When this profile is invoked, the AWS CLI uses the credentials of the `source_profile` to request credentials for the role. Because of that, the identity referenced as the `source_profile` must have `sts:AssumeRole` permissions to the role that is specified in the `role_arn`.

   ```
   [profile prodaccess]
       role_arn = arn:aws:iam::123456789012:role/ProductionAccessRole
       source_profile = default
   ```

3. After you create the new profile, any AWS CLI command that specifies the parameter `--profile prodaccess` runs under the permissions that are attached to the IAM role `ProductionAccessRole` instead of the default user.

   ```
   aws iam list-users --profile prodaccess
   ```

   This command works if the permissions assigned to the `ProductionAccessRole` enable listing the users in the current AWS account.

4. To return to the permissions granted by your original credentials, run commands without the `--profile` parameter. The AWS CLI reverts to using the credentials in your default profile, which you configured in Step 1.

For more information, see Assuming a Role in the *AWS Command Line Interface User Guide*.

Example scenario: Allow an instance profile role to switch to a role in another account

Imagine that you are using two AWS accounts, and you want to allow an application running on an Amazon EC2 instance to run AWS CLI commands in both accounts. Assume that the EC2 instance exists in account `111111111111`. That instance includes the `abcd` instance profile role that allows the application to perform read-only Amazon S3 tasks on the `amzn-s3-demo-bucket1` bucket within the same `111111111111` account. However, the application must also be allowed to assume the `efgh` cross-account role to perform tasks in account `222222222222`. To do this, the `abcd` EC2 instance profile role must have the following permissions policy:

Account 111111111111 abcd role permissions policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccountLevelS3Actions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "AllowListAndReadS3ActionOnMyBucket",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1/*",
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
        },
        {
            "Sid": "AllowIPToAssumeCrossAccountRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::222222222222:role/efgh"
        }
    ]
}
```

Assume that the `efgh` cross-account role allows read-only Amazon S3 tasks on the `amzn-s3-demo-bucket2` bucket within the same `222222222222` account. To do this, the `efgh` cross-account role must have the following permissions policy:

Account 222222222222 efgh role permissions policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccountLevelS3Actions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "AllowListAndReadS3ActionOnMyBucket",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket2/*",
                "arn:aws:s3:::amzn-s3-demo-bucket2"
            ]
        }
    ]
}
```

The `efgh` role must allow the `abcd` instance profile role to assume it. To do this, the `efgh` role must have the following trust policy:

Account 222222222222 efgh role trust policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "efghTrustPolicy",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::111111111111:role/abcd"}
        }
    ]
}
```

To then run AWS CLI commands in account 222222222222, you must update the CLI configuration file. Identify the `efgh` role as the "profile" and the `abcd` EC2 instance profile role as the "credential source" in the AWS CLI configuration file. Then your CLI commands are run with the permissions of the `efgh` role, not the original `abcd` role.

> **Note**
>
> For security purposes, you can use AWS CloudTrail to audit the use of roles in the account. To differentiate between role sessions when a role is used by different principals in CloudTrail logs, you can use the role session name. When the AWS CLI assumes a role on a user's behalf as described in this topic, a role session name is automatically created as `AWS-CLI-session-`*nnnnnnnn*. Here *nnnnnnnn* is an integer that represents the time in Unix epoch time (the number of seconds since midnight UTC on January 1, 1970). For more information, see CloudTrail Event Reference in the *AWS CloudTrail User Guide*.

To allow an EC2 instance profile role to switch to a cross-account role (AWS CLI)

1. You do not have to configure a default CLI profile. Instead, you can load credentials from the EC2 instance profile metadata. Create a new profile for the role in the `.aws/config` file. The following example creates an `instancecrossaccount` profile that switches to the role *`efgh`* in the 222222222222 account. When this profile is invoked, the AWS CLI uses the credentials of the EC2 instance profile metadata to request credentials for the role. Because of that, the

EC2 instance profile role must have `sts:AssumeRole` permissions to the role specified in the `role_arn`.

```
[profile instancecrossaccount]
role_arn = arn:aws:iam::222222222222:role/efgh
credential_source = Ec2InstanceMetadata
```

2. After you create the new profile, any AWS CLI command that specifies the parameter `--profile instancecrossaccount` runs under the permissions that are attached to the `efgh` role in account `222222222222`.

   ```
   aws s3 ls amzn-s3-demo-bucket2 --profile instancecrossaccount
   ```

   This command works if the permissions that are assigned to the `efgh` role allow listing the users in the current AWS account.

3. To return to the original EC2 instance profile permissions in account `111111111111`, run the CLI commands without the `--profile` parameter.

For more information, see Assuming a Role in the *AWS Command Line Interface User Guide*.

Switch to an IAM role (Tools for Windows PowerShell)

A *role* specifies a set of permissions that you can use to access AWS resources that you need. In that sense, it is similar to a user in AWS Identity and Access Management (IAM). When you sign in as a user, you get a specific set of permissions. However, you don't sign in to a role, but once signed in you can switch to a role. This temporarily sets aside your original user permissions and instead gives you the permissions assigned to the role. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create and configure them, see IAM roles, and IAM role creation.

> **Important**
>
> The permissions of your IAM user and any roles that you switch to are not cumulative. Only one set of permissions is active at a time. When you switch to a role, you temporarily give up your user permissions and work with the permissions that are assigned to the role. When you exit the role, your user permissions are automatically restored.

This section describes how to switch roles when you work at the command line with the AWS Tools for Windows PowerShell.

Imagine that you have an account in the development environment and you occasionally need to work with the production environment at the command line using the Tools for Windows PowerShell. You already have one access key credential set available to you. These can be an access key pair assigned to your standard IAM user. Or, if you signed-in as a SAML or OIDC federated principal, they can be the access key pair for the role initially assigned to you. You can use these credentials to run the `Use-STSRole` cmdlet that passes the ARN of a new role as a parameter. The command returns temporary security credentials for the requested role. You can then use those credentials in subsequent PowerShell commands with the role's permissions to access resources in production. While you use the role, you cannot use your user permissions in the Development account because only one set of permissions is in effect at a time.

> **Note**
>
> For security purposes, administrators can review AWS CloudTrail logs to learn who performed an action in AWS. Your administrator might require that you specify a source identity or a role session name when you assume the role. For more information, see `sts:SourceIdentity` and `sts:RoleSessionName`.

Note that all access keys and tokens are examples only and cannot be used as shown. Replace with the appropriate values from your live environment.

To switch to a role (Tools for Windows PowerShell)

1. Open a PowerShell command prompt and configure the default profile to use the access key from your current IAM user or from your federated role. If you have previously used the Tools for Windows PowerShell, then this is likely already done. Note that you can switch roles only if you are signed in as an IAM user, not the AWS account root user.

   ```
   PS C:\> Set-AWSCredentials -AccessKey AKIAIOSFODNN7EXAMPLE -
   SecretKey wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY -StoreAs MyMainUserProfile
   PS C:\> Initialize-AWSDefaults -ProfileName MyMainUserProfile -Region us-east-2
   ```

   For more information, see Using AWS Credentials in the *AWS Tools for PowerShell User Guide*.

2. To retrieve credentials for the new role, run the following command to switch to the *RoleName* role in the 123456789012 account. You get the role ARN from the account administrator who created the role. The command requires that you provide a session name as well. You can choose any text for that. The following command requests the credentials and then captures the `Credentials` property object from the returned results object and stores it in the `$Creds` variable.

```
PS C:\> $Creds = (Use-STSRole -RoleArn "arn:aws:iam::123456789012:role/RoleName" -
RoleSessionName "MyRoleSessionName").Credentials
```

   `$Creds` is an object that now contains the `AccessKeyId`, `SecretAccessKey`, and `SessionToken` elements that you need in the following steps. The following sample commands illustrate typical values:

```
PS C:\> $Creds.AccessKeyId
AKIAIOSFODNN7EXAMPLE

PS C:\> $Creds.SecretAccessKey
wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

PS C:\> $Creds.SessionToken
AQoDYXdzEGcaEXAMPLE2gsYULo
+Im5ZEXAMPLEeYjs1M2FUIgIJx9tQqNMBEXAMPLECvSRyh0FW7jEXAMPLEW+vE/7s1HRp
XviG7b+qYf4nD00EXAMPLEmj4wxS04L/uZEXAMPLECihzFB5lTYLto9dyBgSDyEXAMPLE9/
g7QRUhZp4bqbEXAMPLENwGPy
Oj59pFA4lNKCIkVgkREXAMPLEjlzxQ7y52gekeVEXAMPLEDiB9ST3UuysgsKdEXAMPLE1TVastU1A0SKFEXAMPLEiyw
C
s8EXAMPLEpZgOs+6hz4AP4KEXAMPLERbASP+4eZScEXAMPLEsnf87eNhyDHq6ikBQ==

PS C:\> $Creds.Expiration
Thursday, June 18, 2018 2:28:31 PM
```

3. To use these credentials for any subsequent command, include them with the `-Credential` parameter. For example, the following command uses the credentials from the role and works only if the role is granted the `iam:ListRoles` permission and can therefore run the `Get-IAMRoles` cmdlet:

```
PS C:\> get-iamroles -Credential $Creds
```

4. To return to your original credentials, simply stop using the `-Credentials $Creds` parameter and allow PowerShell to revert to the credentials that are stored in the default profile.

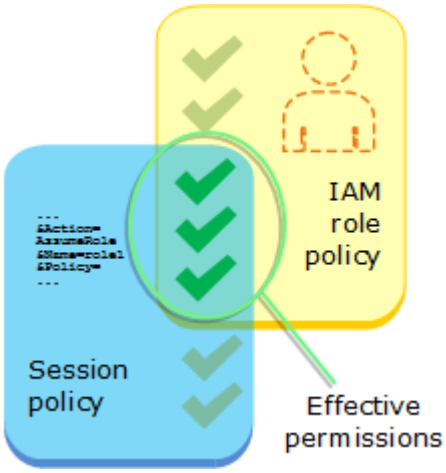
Switch to an IAM role (AWS API)

A *role* specifies a set of permissions that you can use to access AWS resources. In that sense, it is similar to an IAM user. A principal (person or application) assumes a role to receive temporary permissions to carry out required tasks and interact with AWS resources. The role can be in your own account or any other AWS account. For more information about roles, their benefits, and how to create and configure them, see IAM roles, and IAM role creation. To learn about the different methods that you can use to assume a role, see Methods to assume a role.

> **Important**
>
> The permissions of your IAM user and any roles that you assume are not cumulative. Only one set of permissions is active at a time. When you assume a role, you temporarily give up your previous user or role permissions and work with the permissions that are assigned to the role. When you exit the role, your original permissions are automatically restored.

To assume a role, an application calls the AWS STS `AssumeRole` API operation and passes the ARN of the role to use. The operation creates a new session with temporary credentials. This session has the same permissions as the identity-based policies for that role.

When you call `AssumeRole`, you can optionally pass inline or managed session policies. Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary credential session for a role or federated user session. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. The resulting session's permissions are the intersection of the entity's identity-based policies and the session policies. Session policies are useful when you need to give the role's temporary credentials to someone else. They can use the role's temporary credentials in subsequent AWS API calls to access resources in the account that owns the role. You cannot use session policies to grant more permissions than those allowed by the identity-based policy. To learn more about how AWS determines the effective permissions of a role, see Policy evaluation logic.



You can call `AssumeRole` when you are signed in as an IAM user, or as an externally authenticated user (SAML or OIDC) already using a role. You can also use *role chaining*, which is using a role to assume a second role. You cannot assume a role when you are signed in as the AWS account root user.

By default, your role session lasts for one hour. When you assume this role using the AWS STS `AssumeRole*` API operations, you can specify a value for the `DurationSeconds` parameter. This value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see Update the maximum session duration for a role.

If you use role chaining, your session is limited to a maximum of one hour. If you then use the `DurationSeconds` parameter to provide a value greater than one hour, the operation fails.

> **Note**
>
> For security purposes, administrators can review AWS CloudTrail logs to learn who performed an action in AWS. Your administrator might require that you specify a source identity or a role session name when you assume the role. For more information, see `sts:SourceIdentity` and `sts:RoleSessionName`.

The following code examples show how to create a user and assume a role.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;


namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;
```

```
    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
 AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
 CreateAccessKeyRequest
        {
            UserName = userName,
        });
```

```
        return response.AccessKey;

    }


    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</
param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
 policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new
 CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }


    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
 rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
```

```
        return response.Role.Arn;
    }


    /// <summary>
    /// Create an IAM service-linked role.
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName,
 string description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }


    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
 { UserName = userName });
        return response.User;
    }


    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new
DeletePolicyRequest { PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role policy.
```

```
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
 policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
 DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
 { UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
 userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
 DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM
 policy.</param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
 GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }


    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</
param>
    /// <returns>The IAM policy.</returns>
```

```
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }


    /// <summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
            RoleName = roleName,
        });

        return response.Role;
    }


    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest
{ UserName = userName });
        return response.User;
    }


    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
```

```
   public async Task<List<AttachedPolicyType>>
ListAttachedRolePoliciesAsync(string roleName)
   {
       var attachedPolicies = new List<AttachedPolicyType>();
       var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new
ListAttachedRolePoliciesRequest { RoleName = roleName });

       await foreach (var response in attachedRolePoliciesPaginator.Responses)
       {
           attachedPolicies.AddRange(response.AttachedPolicies);
       }

       return attachedPolicies;
   }


   /// <summary>
   /// List IAM groups.
   /// </summary>
   /// <returns>A list of IAM groups.</returns>
   public async Task<List<Group>> ListGroupsAsync()
   {
       var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
       var groups = new List<Group>();

       await foreach (var response in groupsPaginator.Responses)
       {
           groups.AddRange(response.Groups);
       }

       return groups;
   }


   /// <summary>
   /// List IAM policies.
   /// </summary>
   /// <returns>A list of the IAM policies.</returns>
   public async Task<List<ManagedPolicy>> ListPoliciesAsync()
   {
       var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
```

```
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }


    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</
param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator =
 _IAMService.Paginators.ListRolePolicies(new ListRolePoliciesRequest { RoleName =
 roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }


    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
 ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
```

```
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }


    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }


    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
```

```
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string
 roleName, string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM
 policy.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string
 policyName, string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Wait for a new access key to be ready to use.
    /// </summary>
    /// <param name="accessKeyId">The Id of the access key.</param>
```

```
    /// <returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId =
 accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}



using Microsoft.Extensions.Configuration;

namespace IAMBasics;

public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
```

```
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>()
            .AddTransient<IAMWrapper>()
            .AddTransient<UIWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<IAMBasics>();


        IConfiguration configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // Values needed for user, role, and policies.
        string userName = configuration["UserName"]!;
        string s3PolicyName = configuration["S3PolicyName"]!;
        string roleName = configuration["RoleName"]!;


        var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        uiWrapper.DisplayBasicsOverview();
        uiWrapper.PressEnter();

        // First create a user. By default, the new user has
        // no permissions.
        uiWrapper.DisplayTitle("Create User");
        Console.WriteLine($"Creating a new user with user name: {userName}.");
        var user = await iamWrapper.CreateUserAsync(userName);
        var userArn = user.Arn;

        Console.WriteLine($"Successfully created user: {userName} with ARN:
{userArn}.");
```

```
        uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for
use.");

        // Define a role policy document that allows the new user
        // to assume the role.
        string assumeRolePolicyDocument = "{" +
          "\"Version\": \"2012-10-17\"," +
          "\"Statement\": [{" +
              "\"Effect\": \"Allow\"," +
              "\"Principal\": {" +
              $" \"AWS\": \"{userArn}\"" +
              "}," +
              "\"Action\": \"sts:AssumeRole\"" +
          "}]" +
        "}";

        // Permissions to list all buckets.
        string policyDocument = "{" +
            "\"Version\": \"2012-10-17\"," +
            " \"Statement\" : [{" +
                " \"Action\" : [\"s3:ListAllMyBuckets\"]," +
                " \"Effect\" : \"Allow\"," +
                " \"Resource\" : \"*\"" +
            "}]" +
        "}";

        // Create an AccessKey for the user.
        uiWrapper.DisplayTitle("Create access key");
        Console.WriteLine("Now let's create an access key for the new user.");
        var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

        var accessKeyId = accessKey.AccessKeyId;
        var secretAccessKey = accessKey.SecretAccessKey;

        Console.WriteLine($"We have created the access key with Access key id:
{accessKeyId}.");

        Console.WriteLine("Now let's wait until the IAM access key is ready to
use.");
        var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

        // Now try listing the Amazon Simple Storage Service (Amazon S3)
        // buckets. This should fail at this point because the user doesn't
        // have permissions to perform this task.
```

```
        uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
        Console.WriteLine("Now let's try to display a list of the user's Amazon
S3 buckets.");
        var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
        var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

        var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
        var buckets = await s3Wrapper.ListMyBucketsAsync();

        Console.WriteLine(buckets is null
            ? "As expected, the call to list the buckets has returned a null
list."
            : "Something went wrong. This shouldn't have worked.");

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create IAM role");
        Console.WriteLine($"Creating the role: {roleName}");

        // Creating an IAM role to allow listing the S3 buckets. A role name
        // is not case sensitive and must be unique to the account for which it
        // is created.
        var roleArn = await iamWrapper.CreateRoleAsync(roleName,
assumeRolePolicyDocument);

        uiWrapper.PressEnter();

        // Create a policy with permissions to list S3 buckets.
        uiWrapper.DisplayTitle("Create IAM policy");
        Console.WriteLine($"Creating the policy: {s3PolicyName}");
        Console.WriteLine("with permissions to list the Amazon S3 buckets for the
account.");
        var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
policyDocument);

        // Wait 15 seconds for the IAM policy to be available.
        uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

        // Attach the policy to the role you created earlier.
        uiWrapper.DisplayTitle("Attach new IAM policy");
        Console.WriteLine("Now let's attach the policy to the role.");
        await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);
```

```
        // Wait 15 seconds for the role to be updated.
        Console.WriteLine();
        uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

        // Use the AWS Security Token Service (AWS STS) to have the user
        // assume the role we created.
        var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

        // Wait for the new credentials to become valid.
        uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");

        var assumedRoleCredentials = await
s3Wrapper.AssumeS3RoleAsync("temporary-session", roleArn);

        // Try again to list the buckets using the client created with
        // the new user's credentials. This time, it should work.
        var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

        s3Wrapper.UpdateClients(s3Client2, stsClient2);

        buckets = await s3Wrapper.ListMyBucketsAsync();

        uiWrapper.DisplayTitle("List Amazon S3 buckets");
        Console.WriteLine("This time we should have buckets to list.");
        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName} created:
{bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        // Now clean up all the resources used in the example.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Thank you for watching. The IAM Basics demo is
complete.");
        Console.WriteLine("Please wait while we clean up the resources we
created.");

        await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);
```

```
        await iamWrapper.DeletePolicyAsync(policy.Arn);

        await iamWrapper.DeleteRoleAsync(roleName);

        await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

        await iamWrapper.DeleteUserAsync(userName);

        uiWrapper.PressEnter();

        Console.WriteLine("All done cleaning up our resources. Thank you for your
 patience.");
    }
}


namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
```

```
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</
param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
 roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }


    /// <summary>
    /// Delete an S3 bucket.
    /// </summary>
    /// <param name="bucketName">Name of the S3 bucket to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
 { BucketName = bucketName });
        return result.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the buckets that are owned by the user's account.
    /// </summary>
    /// <returns>Async Task.</returns>
    public async Task<List<S3Bucket>?> ListMyBucketsAsync()
    {
        try
        {
            // Get the list of buckets accessible by the new user.
            var response = await _s3Service.ListBucketsAsync();
```

```
            return response.Buckets;
        }
        catch (AmazonS3Exception ex)
        {
            // Something else went wrong. Display the error message.
            Console.WriteLine($"Error: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>
    /// <returns>A Boolean value indicating whether the action completed
    /// successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
 { BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
    /// </summary>
    /// <param name="s3Service">The Amazon S3 client object.</param>
    /// <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
 stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}


namespace IamScenariosCommon;

public class UIWrapper
{
```

```
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
(IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it
full access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    /// <summary>
    /// Show information about the IAM Basics scenario.
    /// </summary>
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries
to list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the
role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
```

```
        Console.WriteLine("\t7. Deletes all the resources.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title, and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);
```

```
        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function iam_create_user_assume_role
#
# Scenario to create an IAM user, create an IAM role, and apply the role to the
 user.
#
#     "IAM access" permissions are needed to run this code.
#     "STS assume role" permissions are needed to run this code. (Note: It might
 be necessary to
#           create a custom policy).
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
##############################################################################
function iam_create_user_assume_role() {
  {
    if [ "$IAM_OPERATIONS_SOURCED" != "True" ]; then

      source ./iam_operations.sh
    fi
  }

  echo_repeat "*" 88
  echo "Welcome to the IAM create user and assume role demo."
  echo
  echo "This demo will create an IAM user, create an IAM role, and apply the role
 to the user."
  echo_repeat "*" 88
  echo

  echo -n "Enter a name for a new IAM user: "
  get_input
  user_name=$get_input_result

  local user_arn
  user_arn=$(iam_create_user -u "$user_name")

  # shellcheck disable=SC2181
  if [[ ${?} == 0 ]]; then
    echo "Created demo IAM user named $user_name"
  else
    errecho "$user_arn"
```

```
    errecho "The user failed to create. This demo will exit."
    return 1
  fi

  local access_key_response
  access_key_response=$(iam_create_user_access_key -u "$user_name")
  # shellcheck disable=SC2181
  if [[ ${?} != 0 ]]; then
    errecho "The access key failed to create. This demo will exit."
    clean_up "$user_name"
    return 1
  fi

  IFS=$'\t ' read -r -a access_key_values <<<"$access_key_response"
  local key_name=${access_key_values[0]}
  local key_secret=${access_key_values[1]}

  echo "Created access key named $key_name"

  echo "Wait 10 seconds for the user to be ready."
  sleep 10
  echo_repeat "*" 88
  echo

  local iam_role_name
  iam_role_name=$(generate_random_name "test-role")
  echo "Creating a role named $iam_role_name with user $user_name as the
principal."

  local assume_role_policy_document="{
    \"Version\": \"2012-10-17\",
    \"Statement\": [{
        \"Effect\": \"Allow\",
        \"Principal\": {\"AWS\": \"$user_arn\"},
        \"Action\": \"sts:AssumeRole\"
        }]
    }"

  local role_arn
  role_arn=$(iam_create_role -n "$iam_role_name" -p
"$assume_role_policy_document")

  # shellcheck disable=SC2181
  if [ ${?} == 0 ]; then
```

```
  echo "Created IAM role named $iam_role_name"
else
  errecho "The role failed to create. This demo will exit."
  clean_up "$user_name" "$key_name"
  return 1
fi

local policy_name
policy_name=$(generate_random_name "test-policy")
local policy_document="{
              \"Version\": \"2012-10-17\",
              \"Statement\": [{
                  \"Effect\": \"Allow\",
                  \"Action\": \"s3:ListAllMyBuckets\",
                  \"Resource\": \"arn:aws:s3:::*\"}]}"

local policy_arn
policy_arn=$(iam_create_policy -n "$policy_name" -p "$policy_document")
# shellcheck disable=SC2181
if [[ ${?} == 0 ]]; then
  echo "Created  IAM policy named $policy_name"
else
  errecho "The policy failed to create."
  clean_up "$user_name" "$key_name" "$iam_role_name"
  return 1
fi

if (iam_attach_role_policy -n "$iam_role_name" -p "$policy_arn"); then
  echo "Attached policy $policy_arn to role $iam_role_name"
else
  errecho "The policy failed to attach."
  clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
  return 1
fi

local assume_role_policy_document="{
              \"Version\": \"2012-10-17\",
              \"Statement\": [{
                  \"Effect\": \"Allow\",
                  \"Action\": \"sts:AssumeRole\",
                  \"Resource\": \"$role_arn\"}]}"

local assume_role_policy_name
assume_role_policy_name=$(generate_random_name "test-assume-role-")
```

```
 # shellcheck disable=SC2181
 local assume_role_policy_arn
 assume_role_policy_arn=$(iam_create_policy -n "$assume_role_policy_name" -p
"$assume_role_policy_document")
 # shellcheck disable=SC2181
 if [ ${?} == 0 ]; then
   echo "Created  IAM policy named $assume_role_policy_name for sts assume role"
 else
   errecho "The policy failed to create."
   clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
"$policy_arn"
   return 1
 fi

 echo "Wait 10 seconds to give AWS time to propagate these new resources and
connections."
 sleep 10
 echo_repeat "*" 88
 echo

 echo "Try to list buckets without the new user assuming the role."
 echo_repeat "*" 88
 echo

 # Set the environment variables for the created user.
 # bashsupport disable=BP2001
 export AWS_ACCESS_KEY_ID=$key_name
 # bashsupport disable=BP2001
 export AWS_SECRET_ACCESS_KEY=$key_secret

 local buckets
 buckets=$(s3_list_buckets)

 # shellcheck disable=SC2181
 if [ ${?} == 0 ]; then
   local bucket_count
   bucket_count=$(echo "$buckets" | wc -w | xargs)
   echo "There are $bucket_count buckets in the account. This should not have
happened."
 else
   errecho "Because the role with permissions has not been assumed, listing
buckets failed."
 fi
```

```
  echo
  echo_repeat "*" 88
  echo "Now assume the role $iam_role_name and list the buckets."
  echo_repeat "*" 88
  echo

  local credentials

  credentials=$(sts_assume_role -r "$role_arn" -n "AssumeRoleDemoSession")
  # shellcheck disable=SC2181
  if [ ${?} == 0 ]; then
    echo "Assumed role $iam_role_name"
  else
    errecho "Failed to assume role."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
"$policy_arn" "$assume_role_policy_arn"
    return 1
  fi

  IFS=$'\t ' read -r -a credentials <<<"$credentials"

  export AWS_ACCESS_KEY_ID=${credentials[0]}
  export AWS_SECRET_ACCESS_KEY=${credentials[1]}
  # bashsupport disable=BP2001
  export AWS_SESSION_TOKEN=${credentials[2]}

  buckets=$(s3_list_buckets)

  # shellcheck disable=SC2181
  if [ ${?} == 0 ]; then
    local bucket_count
    bucket_count=$(echo "$buckets" | wc -w | xargs)
    echo "There are $bucket_count buckets in the account. Listing buckets
succeeded because of "
    echo "the assumed role."
  else
    errecho "Failed to list buckets. This should not happen."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    export AWS_SESSION_TOKEN=""
```

```
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
 "$policy_arn" "$assume_role_policy_arn"
    return 1
  fi

  local result=0
  export AWS_ACCESS_KEY_ID=""
  export AWS_SECRET_ACCESS_KEY=""

  echo
  echo_repeat "*" 88
  echo "The created resources will now be deleted."
  echo_repeat "*" 88
  echo

  clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
 "$assume_role_policy_arn"

  # shellcheck disable=SC2181
  if [[ ${?} -ne 0 ]]; then
    result=1
  fi

  return $result
}
```

The IAM functions used in this scenario.

```
###############################################################################
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
 (IAM) user already exists.
#
# Parameters:
#       $1 - The name of the IAM user to check.
#
# Returns:
#       0 - If the user already exists.
#       1 - If the user doesn't exist.
###############################################################################
function iam_user_exists() {
```

```
  local user_name
  user_name=$1

  # Check whether the IAM user already exists.
  # We suppress all output - we're interested only in the return code.

  local errors
  errors=$(aws iam get-user \
    --user-name "$user_name" 2>&1 >/dev/null)

  local error_code=${?}

  if [[ $error_code -eq 0 ]]; then
    return 0 # 0 in Bash script means true.
  else
    if [[ $errors != *"error"*"(NoSuchEntity)"* ]]; then
      aws_cli_error_log $error_code
      errecho "Error calling iam get-user $errors"
    fi

    return 1 # 1 in Bash script means false.
  fi
}

########################################################################################
# function iam_create_user
#
# This function creates the specified IAM user, unless
# it already exists.
#
# Parameters:
#       -u user_name  -- The name of the user to create.
#
# Returns:
#       The ARN of the user.
#     And:
#       0 - If successful.
#       1 - If it fails.
########################################################################################
function iam_create_user() {
  local user_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
```

```
 function usage() {
   echo "function iam_create_user"
   echo "Creates an AWS Identity and Access Management (IAM) user. You must
supply a username:"
   echo "  -u user_name    The name of the user. It must be unique within the
account."
   echo ""
 }

 # Retrieve the calling parameters.
 while getopts "u:h" option; do
   case "${option}" in
     u) user_name="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$user_name" ]]; then
   errecho "ERROR: You must provide a username with the -u parameter."
   usage
   return 1
 fi

 iecho "Parameters:\n"
 iecho "    User name:   $user_name"
 iecho ""

 # If the user already exists, we don't want to try to create it.
 if (iam_user_exists "$user_name"); then
   errecho "ERROR: A user with that name already exists in the account."
   return 1
 fi

 response=$(aws iam create-user --user-name "$user_name" \
   --output text \
```

```
    --query 'User.Arn')

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-user operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_create_user_access_key
#
# This function creates an IAM access key for the specified user.
#
# Parameters:
#       -u user_name -- The name of the IAM user.
#       [-f file_name] -- The optional file name for the access key output.
#
# Returns:
#       [access_key_id access_key_secret]
#     And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_user_access_key() {
  local user_name file_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user_access_key"
    echo "Creates an AWS Identity and Access Management (IAM) key pair."
    echo "  -u user_name   The name of the IAM user."
    echo "  [-f file_name]   Optional file name for the access key output."
    echo ""
  }

  # Retrieve the calling parameters.
```

```
  while getopts "u:f:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      f) file_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  response=$(aws iam create-access-key \
    --user-name "$user_name" \
    --output text)

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-access-key operation failed.$response"
    return 1
  fi

  if [[ -n "$file_name" ]]; then
    echo "$response" >"$file_name"
  fi

  local key_id key_secret
  # shellcheck disable=SC2086
  key_id=$(echo $response | cut -f 2 -d ' ')
  # shellcheck disable=SC2086
  key_secret=$(echo $response | cut -f 4 -d ' ')
```

```
  echo "$key_id $key_secret"

  return 0
}

###############################################################################
# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_json -- The assume role policy document.
#
# Returns:
#       The ARN of the role.
#     And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_role() {
  local role_name policy_document response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user_access_key"
    echo "Creates an AWS Identity and Access Management (IAM) role."
    echo "  -n role_name   The name of the IAM role."
    echo "  -p policy_json -- The assume role policy document."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:p:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      p) policy_document="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
```

```
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$policy_document" ]]; then
    errecho "ERROR: You must provide a policy document with the -p parameter."
    usage
    return 1
  fi

  response=$(aws iam create-role \
    --role-name "$role_name" \
    --assume-role-policy-document "$policy_document" \
    --output text \
    --query Role.Arn)

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_create_policy
#
# This function creates an IAM policy.
#
```

```
# Parameters:
#       -n policy_name -- The name of the IAM policy.
#       -p policy_json -- The policy document.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_policy() {
  local policy_name policy_document response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_policy"
    echo "Creates an AWS Identity and Access Management (IAM) policy."
    echo "  -n policy_name   The name of the IAM policy."
    echo "  -p policy_json -- The policy document."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:p:h" option; do
    case "${option}" in
      n) policy_name="${OPTARG}" ;;
      p) policy_document="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$policy_name" ]]; then
    errecho "ERROR: You must provide a policy name with the -n parameter."
    usage
    return 1
  fi
```

```
  if [[ -z "$policy_document" ]]; then
    errecho "ERROR: You must provide a policy document with the -p parameter."
    usage
    return 1
  fi

  response=$(aws iam create-policy \
    --policy-name "$policy_name" \
    --policy-document "$policy_document" \
    --output text \
    --query Policy.Arn)

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-policy operation failed.\n$response"
    return 1
  fi

  echo "$response"
}

###############################################################################
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a tole.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_attach_role_policy() {
  local role_name policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_attach_role_policy"
```

```
    echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM
role."
    echo "  -n role_name   The name of the IAM role."
    echo "  -p policy_ARN -- The IAM policy document ARN."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:p:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      p) policy_arn="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy ARN with the -p parameter."
    usage
    return 1
  fi

  response=$(aws iam attach-role-policy \
    --role-name "$role_name" \
    --policy-arn "$policy_arn")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
```

```
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a tole.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_detach_role_policy() {
  local role_name policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_detach_role_policy"
    echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM
 role."
    echo "  -n role_name   The name of the IAM role."
    echo "  -p policy_ARN -- The IAM policy document ARN."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:p:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      p) policy_arn="${OPTARG}" ;;
      h)
        usage
```

```
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy ARN with the -p parameter."
    usage
    return 1
  fi

  response=$(aws iam detach-role-policy \
    --role-name "$role_name" \
    --policy-arn "$policy_arn")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_delete_policy
#
# This function deletes an IAM policy.
```

```
#
# Parameters:
#       -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_policy() {
  local policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_policy"
    echo "Deletes an AWS Identity and Access Management (IAM) policy"
    echo "  -n policy_arn -- The name of the IAM policy arn."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) policy_arn="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy arn with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
```

```
  iecho "    Policy arn:  $policy_arn"
  iecho ""

  response=$(aws iam delete-policy \
    --policy-arn "$policy_arn")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
  fi

  iecho "delete-policy response:$response"
  iecho

  return 0
}

###############################################################################
# function iam_delete_role
#
# This function deletes an IAM role.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_role() {
  local role_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_role"
    echo "Deletes an AWS Identity and Access Management (IAM) role"
    echo "  -n role_name -- The name of the IAM role."
    echo ""
  }
```

```
  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  echo "role_name:$role_name"
  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    Role name:  $role_name"
  iecho ""

  response=$(aws iam delete-role \
    --role-name "$role_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
  fi

  iecho "delete-role response:$response"
  iecho

  return 0
}
```

```
###############################################################################
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#       -u user_name  -- The name of the user.
#       -k access_key -- The access key to delete.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_access_key() {
  local user_name access_key response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_access_key"
    echo "Deletes an AWS Identity and Access Management (IAM) access key for the
 specified IAM user"
    echo "  -u user_name    The name of the user."
    echo "  -k access_key   The access key to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "u:k:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      k) access_key="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
```

```
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    Username:   $user_name"
  iecho "    Access key:   $access_key"
  iecho ""

  response=$(aws iam delete-access-key \
    --user-name "$user_name" \
    --access-key-id "$access_key")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
    return 1
  fi

  iecho "delete-access-key response:$response"
  iecho

  return 0
}

########################################################################################
# function iam_delete_user
#
# This function deletes the specified IAM user.
#
# Parameters:
#       -u user_name  -- The name of the user to create.
```

```
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_user() {
  local user_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_user"
    echo "Deletes an AWS Identity and Access Management (IAM) user. You must
 supply a username:"
    echo "  -u user_name    The name of the user."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "u:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    User name:   $user_name"
  iecho ""
```

```
  # If the user does not exist, we don't want to try to delete it.
  if (! iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name does not exist in the account."
    return 1
  fi

  response=$(aws iam delete-user \
    --user-name "$user_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-user operation failed.$response"
    return 1
  fi

  iecho "delete-user response:$response"
  iecho

  return 0
}
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace AwsDoc {
    namespace IAM {

        //! Cleanup by deleting created entities.
        /*!
          \sa DeleteCreatedEntities
          \param client: IAM client.
          \param role: IAM role.
          \param user: IAM user.
          \param policy: IAM policy.
        */
        static bool DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
                                          const Aws::IAM::Model::Role &role,
                                          const Aws::IAM::Model::User &user,
                                          const Aws::IAM::Model::Policy &policy);
    }

    static const int LIST_BUCKETS_WAIT_SEC = 20;

    static const char ALLOCATION_TAG[] = "example_code";
}

//! Scenario to create an IAM user, create an IAM role, and apply the role to the
 user.
// "IAM access" permissions are needed to run this code.
// "STS assume role" permissions are needed to run this code. (Note: It might be
 necessary to
//    create a custom policy).
/*!
  \sa iamCreateUserAssumeRoleScenario
  \param clientConfig: Aws client configuration.
  \return bool: Successful completion.
```

```
*/
bool AwsDoc::IAM::iamCreateUserAssumeRoleScenario(
        const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::IAM::IAMClient client(clientConfig);
    Aws::IAM::Model::User user;
    Aws::IAM::Model::Role role;
    Aws::IAM::Model::Policy policy;

    // 1. Create a user.
    {
        Aws::IAM::Model::CreateUserRequest request;
        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String userName = "iam-demo-user-" +
                               Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetUserName(userName);

        Aws::IAM::Model::CreateUserOutcome outcome = client.CreateUser(request);
        if (!outcome.IsSuccess()) {
            std::cout << "Error creating IAM user " << userName << ":" <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            std::cout << "Successfully created IAM user " << userName <<
 std::endl;
        }

        user = outcome.GetResult().GetUser();
    }

    // 2. Create a role.
    {
        // Get the IAM user for the current client in order to access its ARN.
        Aws::String iamUserArn;
        {
            Aws::IAM::Model::GetUserRequest request;
            Aws::IAM::Model::GetUserOutcome outcome = client.GetUser(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Error getting Iam user. " <<
                          outcome.GetError().GetMessage() << std::endl;

                DeleteCreatedEntities(client, role, user, policy);
                return false;
```

```
            }
            else {
                std::cout << "Successfully retrieved Iam user "
                          << outcome.GetResult().GetUser().GetUserName()
                          << std::endl;
            }

            iamUserArn = outcome.GetResult().GetUser().GetArn();
        }

        Aws::IAM::Model::CreateRoleRequest request;

        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String roleName = "iam-demo-role-" +
                               Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetRoleName(roleName);

        // Build policy document for role.
        Aws::Utils::Document jsonStatement;
        jsonStatement.WithString("Effect", "Allow");

        Aws::Utils::Document jsonPrincipal;
        jsonPrincipal.WithString("AWS", iamUserArn);
        jsonStatement.WithObject("Principal", jsonPrincipal);
        jsonStatement.WithString("Action", "sts:AssumeRole");
        jsonStatement.WithObject("Condition", Aws::Utils::Document());

        Aws::Utils::Document policyDocument;
        policyDocument.WithString("Version", "2012-10-17");

        Aws::Utils::Array<Aws::Utils::Document> statements(1);
        statements[0] = jsonStatement;
        policyDocument.WithArray("Statement", statements);

        std::cout << "Setting policy for role\n   "
                  << policyDocument.View().WriteCompact() << std::endl;

        // Set role policy document as JSON string.

    request.SetAssumeRolePolicyDocument(policyDocument.View().WriteCompact());

        Aws::IAM::Model::CreateRoleOutcome outcome = client.CreateRole(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error creating role. " <<
```

```
                    outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully created a role with name " << roleName
                      << std::endl;
        }

        role = outcome.GetResult().GetRole();
    }

    // 3. Create an IAM policy.
    {
        Aws::IAM::Model::CreatePolicyRequest request;
        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String policyName = "iam-demo-policy-" +
                                 Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetPolicyName(policyName);

        // Build IAM policy document.
        Aws::Utils::Document jsonStatement;
        jsonStatement.WithString("Effect", "Allow");
        jsonStatement.WithString("Action", "s3:ListAllMyBuckets");
        jsonStatement.WithString("Resource", "arn:aws:s3:::*");

        Aws::Utils::Document policyDocument;
        policyDocument.WithString("Version", "2012-10-17");

        Aws::Utils::Array<Aws::Utils::Document> statements(1);
        statements[0] = jsonStatement;
        policyDocument.WithArray("Statement", statements);

        std::cout << "Creating a policy.\n   " <<
policyDocument.View().WriteCompact()
                  << std::endl;

        // Set IAM policy document as JSON string.
        request.SetPolicyDocument(policyDocument.View().WriteCompact());

        Aws::IAM::Model::CreatePolicyOutcome outcome =
client.CreatePolicy(request);
        if (!outcome.IsSuccess()) {
```

```
            std::cerr << "Error creating policy. " <<
                      outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully created a policy with name, " <<
policyName <<
                      "." << std::endl;
        }

        policy = outcome.GetResult().GetPolicy();
    }

    // 4. Assume the new role using the AWS Security Token Service (STS).
    Aws::STS::Model::Credentials credentials;
    {
        Aws::STS::STSClient stsClient(clientConfig);

        Aws::STS::Model::AssumeRoleRequest request;
        request.SetRoleArn(role.GetArn());
        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String roleSessionName = "iam-demo-role-session-" +

Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetRoleSessionName(roleSessionName);

        Aws::STS::Model::AssumeRoleOutcome assumeRoleOutcome;

        // Repeatedly call AssumeRole, because there is often a delay
        // before the role is available to be assumed.
        // Repeat at most 20 times when access is denied.
        int count = 0;
        while (true) {
            assumeRoleOutcome = stsClient.AssumeRole(request);
            if (!assumeRoleOutcome.IsSuccess()) {
                if (count > 20 ||
                    assumeRoleOutcome.GetError().GetErrorType() !=
                    Aws::STS::STSErrors::ACCESS_DENIED) {
                    std::cerr << "Error assuming role after 20 tries. " <<
                              assumeRoleOutcome.GetError().GetMessage() <<
std::endl;
```

```
                    DeleteCreatedEntities(client, role, user, policy);
                    return false;
                }
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                std::cout << "Successfully assumed the role after " << count
                          << " seconds." << std::endl;
                break;
            }
            count++;
        }

        credentials = assumeRoleOutcome.GetResult().GetCredentials();
    }


    // 5. List objects in the bucket (This should fail).
    {
        Aws::S3::S3Client s3Client(
                Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
                                          credentials.GetSecretAccessKey(),
                                          credentials.GetSessionToken()),
                Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
                clientConfig);
        Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
s3Client.ListBuckets();
        if (!listBucketsOutcome.IsSuccess()) {
            if (listBucketsOutcome.GetError().GetErrorType() !=
                Aws::S3::S3Errors::ACCESS_DENIED) {
                std::cerr << "Could not lists buckets. " <<
                          listBucketsOutcome.GetError().GetMessage() <<
std::endl;
            }
            else {
                std::cout
                        << "Access to list buckets denied because privileges have
not been applied."
                        << std::endl;
            }
        }
        else {
            std::cerr
```

```
                    << "Successfully retrieved bucket lists when this should not
happen."
                    << std::endl;
        }
    }

    // 6. Attach the policy to the role.
    {
        Aws::IAM::Model::AttachRolePolicyRequest request;
        request.SetRoleName(role.GetRoleName());
        request.WithPolicyArn(policy.GetArn());

        Aws::IAM::Model::AttachRolePolicyOutcome outcome =
client.AttachRolePolicy(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error creating policy. " <<
                      outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully attached the policy with name, "
                      << policy.GetPolicyName() <<
                      ", to the role, " << role.GetRoleName() << "." <<
std::endl;
        }
    }

    int count = 0;
    // 7. List objects in the bucket (this should succeed).
    // Repeatedly call ListBuckets, because there is often a delay
    // before the policy with ListBucket permissions has been applied to the
role.
    // Repeat at most LIST_BUCKETS_WAIT_SEC times when access is denied.
    while (true) {
        Aws::S3::S3Client s3Client(
                Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
                                          credentials.GetSecretAccessKey(),
                                          credentials.GetSessionToken()),
                Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
                clientConfig);
```

```
        Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
 s3Client.ListBuckets();
        if (!listBucketsOutcome.IsSuccess()) {
            if ((count > LIST_BUCKETS_WAIT_SEC) ||
                listBucketsOutcome.GetError().GetErrorType() !=
                Aws::S3::S3Errors::ACCESS_DENIED) {
                std::cerr << "Could not lists buckets after " <<
 LIST_BUCKETS_WAIT_SEC << " seconds. " <<
                          listBucketsOutcome.GetError().GetMessage() <<
 std::endl;
                DeleteCreatedEntities(client, role, user, policy);
                return false;
            }

            std::this_thread::sleep_for(std::chrono::seconds(1));
        }
        else {

            std::cout << "Successfully retrieved bucket lists after " << count
                      << " seconds." << std::endl;
            break;
        }
        count++;
    }

    // 8. Delete all the created resources.
    return DeleteCreatedEntities(client, role, user, policy);
}

bool AwsDoc::IAM::DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
                                        const Aws::IAM::Model::Role &role,
                                        const Aws::IAM::Model::User &user,
                                        const Aws::IAM::Model::Policy &policy) {
    bool result = true;
    if (policy.ArnHasBeenSet()) {
        // Detach the policy from the role.
        {
            Aws::IAM::Model::DetachRolePolicyRequest request;
            request.SetPolicyArn(policy.GetArn());
            request.SetRoleName(role.GetRoleName());

            Aws::IAM::Model::DetachRolePolicyOutcome outcome =
 client.DetachRolePolicy(
                    request);
```

```
            if (!outcome.IsSuccess()) {
                std::cerr << "Error Detaching policy from roles. " <<
                          outcome.GetError().GetMessage() << std::endl;
                result = false;
            }
            else {
                std::cout << "Successfully detached the policy with arn "
                          << policy.GetArn()
                          << " from role " << role.GetRoleName() << "." <<
std::endl;
            }
        }

        // Delete the policy.
        {
            Aws::IAM::Model::DeletePolicyRequest request;
            request.WithPolicyArn(policy.GetArn());

            Aws::IAM::Model::DeletePolicyOutcome outcome =
client.DeletePolicy(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Error deleting policy. " <<
                          outcome.GetError().GetMessage() << std::endl;
                result = false;
            }
            else {
                std::cout << "Successfully deleted the policy with arn "
                          << policy.GetArn() << std::endl;
            }
        }

    }

    if (role.RoleIdHasBeenSet()) {
        // Delete the role.
        Aws::IAM::Model::DeleteRoleRequest request;
        request.SetRoleName(role.GetRoleName());

        Aws::IAM::Model::DeleteRoleOutcome outcome = client.DeleteRole(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error deleting role. " <<
                      outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
```

```
        else {
            std::cout << "Successfully deleted the role with name "
                      << role.GetRoleName() << std::endl;
        }
    }

    if (user.ArnHasBeenSet()) {
        // Delete the user.
        Aws::IAM::Model::DeleteUserRequest request;
        request.WithUserName(user.GetUserName());

        Aws::IAM::Model::DeleteUserOutcome outcome = client.DeleteUser(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error deleting user. " <<
                      outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Successfully deleted the user with name "
                      << user.GetUserName() << std::endl;
        }
    }

    return result;
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy

- PutUserPolicy

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "math/rand"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/credentials"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/sts"
 "github.com/aws/smithy-go"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/iam/actions"
)

// AssumeRoleScenario shows you how to use the AWS Identity and Access Management
 (IAM)
// service to perform the following actions:
//
//  1. Create a user who has no permissions.
//  2. Create a role that grants permission to list Amazon Simple Storage Service
```

```
//     (Amazon S3) buckets for the account.
//  3. Add a policy to let the user assume the role.
//  4. Try and fail to list buckets without permissions.
//  5. Assume the role and list S3 buckets using temporary credentials.
//  6. Delete the policy, role, and user.
type AssumeRoleScenario struct {
 sdkConfig      aws.Config
 accountWrapper actions.AccountWrapper
 policyWrapper  actions.PolicyWrapper
 roleWrapper    actions.RoleWrapper
 userWrapper    actions.UserWrapper
 questioner     demotools.IQuestioner
 helper         IScenarioHelper
 isTestRun      bool
}

// NewAssumeRoleScenario constructs an AssumeRoleScenario instance from a
 configuration.
// It uses the specified config to get an IAM client and create wrappers for the
 actions
// used in the scenario.
func NewAssumeRoleScenario(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) AssumeRoleScenario {
 iamClient := iam.NewFromConfig(sdkConfig)
 return AssumeRoleScenario{
  sdkConfig:      sdkConfig,
  accountWrapper: actions.AccountWrapper{IamClient: iamClient},
  policyWrapper:  actions.PolicyWrapper{IamClient: iamClient},
  roleWrapper:    actions.RoleWrapper{IamClient: iamClient},
  userWrapper:    actions.UserWrapper{IamClient: iamClient},
  questioner:     questioner,
  helper:         helper,
 }
}

// addTestOptions appends the API options specified in the original configuration
 to
// another configuration. This is used to attach the middleware stubber to
 clients
// that are constructed during the scenario, which is needed for unit testing.
func (scenario AssumeRoleScenario) addTestOptions(scenarioConfig *aws.Config) {
 if scenario.isTestRun {
```

```
  scenarioConfig.APIOptions = append(scenarioConfig.APIOptions,
 scenario.sdkConfig.APIOptions...)
 }
}

// Run runs the interactive scenario.
func (scenario AssumeRoleScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo.\n")
   log.Println(r)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the AWS Identity and Access Management (IAM) assume role
 demo.")
 log.Println(strings.Repeat("-", 88))

 user := scenario.CreateUser(ctx)
 accessKey := scenario.CreateAccessKey(ctx, user)
 role := scenario.CreateRoleAndPolicies(ctx, user)
 noPermsConfig := scenario.ListBucketsWithoutPermissions(ctx, accessKey)
 scenario.ListBucketsWithAssumedRole(ctx, noPermsConfig, role)
 scenario.Cleanup(ctx, user, role)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateUser creates a new IAM user. This user has no permissions.
func (scenario AssumeRoleScenario) CreateUser(ctx context.Context) *types.User {
 log.Println("Let's create an example user with no permissions.")
 userName := scenario.questioner.Ask("Enter a name for the example user:",
 demotools.NotEmpty{})
 user, err := scenario.userWrapper.GetUser(ctx, userName)
 if err != nil {
  panic(err)
 }
 if user == nil {
  user, err = scenario.userWrapper.CreateUser(ctx, userName)
  if err != nil {
   panic(err)
```

```
  }
  log.Printf("Created user %v.\n", *user.UserName)
 } else {
  log.Printf("User %v already exists.\n", *user.UserName)
 }
 log.Println(strings.Repeat("-", 88))
 return user
}

// CreateAccessKey creates an access key for the user.
func (scenario AssumeRoleScenario) CreateAccessKey(ctx context.Context, user
 *types.User) *types.AccessKey {
 accessKey, err := scenario.userWrapper.CreateAccessKeyPair(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("Created access key %v for your user.", *accessKey.AccessKeyId)
 log.Println("Waiting a few seconds for your user to be ready...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return accessKey
}

// CreateRoleAndPolicies creates a policy that grants permission to list S3
 buckets for
// the current account and attaches the policy to a newly created role. It also
 adds an
// inline policy to the specified user that grants the user permission to assume
 the role.
func (scenario AssumeRoleScenario) CreateRoleAndPolicies(ctx context.Context,
 user *types.User) *types.Role {
 log.Println("Let's create a role and policy that grant permission to list S3
 buckets.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 listBucketsRole, err := scenario.roleWrapper.CreateRole(ctx,
 scenario.helper.GetName(), *user.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created role %v.\n", *listBucketsRole.RoleName)
 listBucketsPolicy, err := scenario.policyWrapper.CreatePolicy(
  ctx, scenario.helper.GetName(), []string{"s3:ListAllMyBuckets"},
 "arn:aws:s3:::*")
 if err != nil {
```

```
  panic(err)
 }
 log.Printf("Created policy %v.\n", *listBucketsPolicy.PolicyName)
 err = scenario.roleWrapper.AttachRolePolicy(ctx, *listBucketsPolicy.Arn,
 *listBucketsRole.RoleName)
 if err != nil {
  panic(err)
 }
 log.Printf("Attached policy %v to role %v.\n", *listBucketsPolicy.PolicyName,
  *listBucketsRole.RoleName)
 err = scenario.userWrapper.CreateUserPolicy(ctx, *user.UserName,
 scenario.helper.GetName(),
  []string{"sts:AssumeRole"}, *listBucketsRole.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created an inline policy for user %v that lets the user assume the
 role.\n",
  *user.UserName)
 log.Println("Let's give AWS a few seconds to propagate these new resources and
 connections...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return listBucketsRole
}

// ListBucketsWithoutPermissions creates an Amazon S3 client from the user's
 access key
// credentials and tries to list buckets for the account. Because the user does
 not have
// permission to perform this action, the action fails.
func (scenario AssumeRoleScenario) ListBucketsWithoutPermissions(ctx
 context.Context, accessKey *types.AccessKey) *aws.Config {
 log.Println("Let's try to list buckets without permissions. This should return
 an AccessDenied error.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 noPermsConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *accessKey.AccessKeyId, *accessKey.SecretAccessKey, ""),
  ))
 if err != nil {
  panic(err)
 }
```

```
 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&noPermsConfig)

 s3Client := s3.NewFromConfig(noPermsConfig)
 _, err = s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  // The SDK for Go does not model the AccessDenied error, so check ErrorCode
 directly.
  var ae smithy.APIError
  if errors.As(err, &ae) {
   switch ae.ErrorCode() {
   case "AccessDenied":
    log.Println("Got AccessDenied error, which is the expected result because\n"
 +
     "the ListBuckets call was made without permissions.")
   default:
    log.Println("Expected AccessDenied, got something else.")
    panic(err)
   }
  }
 } else {
  log.Println("Expected AccessDenied error when calling ListBuckets without
 permissions,\n" +
   "but the call succeeded. Continuing the example anyway...")
 }
 log.Println(strings.Repeat("-", 88))
 return &noPermsConfig
}

// ListBucketsWithAssumedRole performs the following actions:
//
//  1. Creates an AWS Security Token Service (AWS STS) client from the config
 created from
//     the user's access key credentials.
//  2. Gets temporary credentials by assuming the role that grants permission to
 list the
//     buckets.
//  3. Creates an Amazon S3 client from the temporary credentials.
//  4. Lists buckets for the account. Because the temporary credentials are
 generated by
//     assuming the role that grants permission, the action succeeds.
func (scenario AssumeRoleScenario) ListBucketsWithAssumedRole(ctx
 context.Context, noPermsConfig *aws.Config, role *types.Role) {
```

```
 log.Println("Let's assume the role that grants permission to list buckets and
 try again.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 stsClient := sts.NewFromConfig(*noPermsConfig)
 tempCredentials, err := stsClient.AssumeRole(ctx, &sts.AssumeRoleInput{
  RoleArn:         role.Arn,
  RoleSessionName: aws.String("AssumeRoleExampleSession"),
  DurationSeconds: aws.Int32(900),
 })
 if err != nil {
  log.Printf("Couldn't assume role %v.\n", *role.RoleName)
  panic(err)
 }
 log.Printf("Assumed role %v, got temporary credentials.\n", *role.RoleName)
 assumeRoleConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *tempCredentials.Credentials.AccessKeyId,
   *tempCredentials.Credentials.SecretAccessKey,
   *tempCredentials.Credentials.SessionToken),
  ),
 )
 if err != nil {
  panic(err)
 }

 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&assumeRoleConfig)

 s3Client := s3.NewFromConfig(assumeRoleConfig)
 result, err := s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  log.Println("Couldn't list buckets with assumed role credentials.")
  panic(err)
 }
 log.Println("Successfully called ListBuckets with assumed role credentials, \n"
 +
  "here are some of them:")
 for i := 0; i < len(result.Buckets) && i < 5; i++ {
  log.Printf("\t%v\n", *result.Buckets[i].Name)
 }
 log.Println(strings.Repeat("-", 88))
}
```

```
// Cleanup deletes all resources created for the scenario.
func (scenario AssumeRoleScenario) Cleanup(ctx context.Context, user *types.User,
 role *types.Role) {
 if scenario.questioner.AskBool(
  "Do you want to delete the resources created for this example? (y/n)", "y",
 ) {
  policies, err := scenario.roleWrapper.ListAttachedRolePolicies(ctx,
 *role.RoleName)
  if err != nil {
   panic(err)
  }
  for _, policy := range policies {
   err = scenario.roleWrapper.DetachRolePolicy(ctx, *role.RoleName,
 *policy.PolicyArn)
   if err != nil {
    panic(err)
   }
   err = scenario.policyWrapper.DeletePolicy(ctx, *policy.PolicyArn)
   if err != nil {
    panic(err)
   }
   log.Printf("Detached policy %v from role %v and deleted the policy.\n",
    *policy.PolicyName, *role.RoleName)
  }
  err = scenario.roleWrapper.DeleteRole(ctx, *role.RoleName)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted role %v.\n", *role.RoleName)

  userPols, err := scenario.userWrapper.ListUserPolicies(ctx, *user.UserName)
  if err != nil {
   panic(err)
  }
  for _, userPol := range userPols {
   err = scenario.userWrapper.DeleteUserPolicy(ctx, *user.UserName, userPol)
   if err != nil {
    panic(err)
   }
   log.Printf("Deleted policy %v from user %v.\n", userPol, *user.UserName)
  }
  keys, err := scenario.userWrapper.ListAccessKeys(ctx, *user.UserName)
  if err != nil {
   panic(err)
```

```
  }
  for _, key := range keys {
   err = scenario.userWrapper.DeleteAccessKey(ctx, *user.UserName,
 *key.AccessKeyId)
   if err != nil {
    panic(err)
   }
   log.Printf("Deleted access key %v from user %v.\n", *key.AccessKeyId,
 *user.UserName)
  }
  err = scenario.userWrapper.DeleteUser(ctx, *user.UserName)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted user %v.\n", *user.UserName)
  log.Println(strings.Repeat("-", 88))
 }

}

// IScenarioHelper abstracts input and wait functions from a scenario so that
 they
// can be mocked for unit testing.
type IScenarioHelper interface {
 GetName() string
 Pause(secs int)
}

const rMax = 100000

type ScenarioHelper struct {
 Prefix string
 Random *rand.Rand
}

// GetName returns a unique name formed of a prefix and a random number.
func (helper *ScenarioHelper) GetName() string {
 return fmt.Sprintf("%v%v", helper.Prefix, helper.Random.Intn(rMax))
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}
```

Define a struct that wraps account actions.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// GetAccountPasswordPolicy gets the account password policy for the current
 account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy(ctx context.Context)
 (*types.PasswordPolicy, error) {
 var pwPolicy *types.PasswordPolicy
 result, err := wrapper.IamClient.GetAccountPasswordPolicy(ctx,
  &iam.GetAccountPasswordPolicyInput{})
 if err != nil {
  log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
 } else {
  pwPolicy = result.PasswordPolicy
 }
 return pwPolicy, err
}



// ListSAMLProviders gets the SAML providers for the account.
```

```
func (wrapper AccountWrapper) ListSAMLProviders(ctx context.Context)
 ([]types.SAMLProviderListEntry, error) {
 var providers []types.SAMLProviderListEntry
 result, err := wrapper.IamClient.ListSAMLProviders(ctx,
 &iam.ListSAMLProvidersInput{})
 if err != nil {
  log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
 } else {
  providers = result.SAMLProviderList
 }
 return providers, err
}
```

Define a struct that wraps policy actions.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(ctx context.Context, maxPolicies int32)
 ([]types.Policy, error) {
 var policies []types.Policy
 result, err := wrapper.IamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
```

```
  MaxItems: aws.Int32(maxPolicies),
 })
 if err != nil {
  log.Printf("Couldn't list policies. Here's why: %v\n", err)
 } else {
  policies = result.Policies
 }
 return policies, err
}



// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    []string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
}

// CreatePolicy creates a policy that grants a list of actions to the specified
 resource.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(ctx context.Context, policyName string,
 actions []string,
 resourceArn string) (*types.Policy, error) {
 var policy *types.Policy
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(resourceArn),
  }},
 }
```

```
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
 resourceArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreatePolicy(ctx, &iam.CreatePolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
 })
 if err != nil {
  log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}



// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(ctx context.Context, policyArn string)
 (*types.Policy, error) {
 var policy *types.Policy
 result, err := wrapper.IamClient.GetPolicy(ctx, &iam.GetPolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}



// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(ctx context.Context, policyArn string)
 error {
 _, err := wrapper.IamClient.DeletePolicy(ctx, &iam.DeletePolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
```

```
  log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
 }
 return err
}
```

Define a struct that wraps role actions.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(ctx context.Context, maxRoles int32)
 ([]types.Role, error) {
 var roles []types.Role
 result, err := wrapper.IamClient.ListRoles(ctx,
  &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
 )
 if err != nil {
  log.Printf("Couldn't list roles. Here's why: %v\n", err)
 } else {
  roles = result.Roles
 }
 return roles, err
}
```

```
// CreateRole creates a role that trusts a specified user. The trusted user can
 assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(ctx context.Context, roleName string,
 trustedUserArn string) (*types.Role, error) {
 var role *types.Role
 trustPolicy := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Principal: map[string]string{"AWS": trustedUserArn},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyBytes, err := json.Marshal(trustPolicy)
 if err != nil {
  log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
 trustedUserArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreateRole(ctx, &iam.CreateRoleInput{
  AssumeRolePolicyDocument: aws.String(string(policyBytes)),
  RoleName:                 aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(ctx context.Context, roleName string)
 (*types.Role, error) {
 var role *types.Role
```

```
 result, err := wrapper.IamClient.GetRole(ctx,
  &iam.GetRoleInput{RoleName: aws.String(roleName)})
 if err != nil {
  log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// CreateServiceLinkedRole creates a service-linked role that is owned by the
 specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(ctx context.Context,
 serviceName string, description string) (
 *types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.CreateServiceLinkedRole(ctx,
 &iam.CreateServiceLinkedRoleInput{
  AWSServiceName: aws.String(serviceName),
  Description:    aws.String(description),
 })
 if err != nil {
  log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
 serviceName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(ctx context.Context, roleName
 string) error {
 _, err := wrapper.IamClient.DeleteServiceLinkedRole(ctx,
 &iam.DeleteServiceLinkedRoleInput{
  RoleName: aws.String(roleName)},
 )
 if err != nil {
  log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n",
 roleName, err)
```

```
 }
 return err
}



// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(ctx context.Context, policyArn
 string, roleName string) error {
 _, err := wrapper.IamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
 roleName, err)
 }
 return err
}



// ListAttachedRolePolicies lists the policies that are attached to the specified
 role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(ctx context.Context, roleName
 string) ([]types.AttachedPolicy, error) {
 var policies []types.AttachedPolicy
 result, err := wrapper.IamClient.ListAttachedRolePolicies(ctx,
 &iam.ListAttachedRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
 roleName, err)
 } else {
  policies = result.AttachedPolicies
 }
 return policies, err
}



// DetachRolePolicy detaches a policy from a role.
```

```
func (wrapper RoleWrapper) DetachRolePolicy(ctx context.Context, roleName string,
 policyArn string) error {
 _, err := wrapper.IamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName,
 err)
 }
 return err
}



// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(ctx context.Context, roleName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListRolePolicies(ctx,
 &iam.ListRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName,
 err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}



// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(ctx context.Context, roleName string) error
 {
 _, err := wrapper.IamClient.DeleteRole(ctx, &iam.DeleteRoleInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
 }
```

```
 return err
}
```

Define a struct that wraps user actions.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(ctx context.Context, maxUsers int32)
 ([]types.User, error) {
 var users []types.User
 result, err := wrapper.IamClient.ListUsers(ctx, &iam.ListUsersInput{
  MaxItems: aws.Int32(maxUsers),
 })
 if err != nil {
  log.Printf("Couldn't list users. Here's why: %v\n", err)
 } else {
  users = result.Users
 }
 return users, err
}
```

```
// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.GetUser(ctx, &iam.GetUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NoSuchEntityException:
    log.Printf("User %v does not exist.\n", userName)
    err = nil
   default:
    log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
   }
  }
 } else {
  user = result.User
 }
 return user, err
}



// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.CreateUser(ctx, &iam.CreateUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
 } else {
  user = result.User
 }
 return user, err
}
```

```
// CreateUserPolicy adds an inline policy to a user. This example creates a
 policy that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(ctx context.Context, userName string,
 policyName string, actions []string,
 roleArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(roleArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
 err)
  return err
 }
 _, err = wrapper.IamClient.PutUserPolicy(ctx, &iam.PutUserPolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
  UserName:       aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName,
 err)
 }
 return err
}



// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(ctx context.Context, userName string)
 ([]string, error) {
 var policies []string
```

```
 result, err := wrapper.IamClient.ListUserPolicies(ctx,
 &iam.ListUserPoliciesInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName,
 err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}



// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(ctx context.Context, userName string,
 policyName string) error {
 _, err := wrapper.IamClient.DeleteUserPolicy(ctx, &iam.DeleteUserPolicyInput{
  PolicyName: aws.String(policyName),
  UserName:   aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName,
 err)
 }
 return err
}



// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(ctx context.Context, userName string) error
 {
 _, err := wrapper.IamClient.DeleteUser(ctx, &iam.DeleteUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

```
// CreateAccessKeyPair creates an access key for a user. The returned access key
 contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(ctx context.Context, userName
 string) (*types.AccessKey, error) {
 var key *types.AccessKey
 result, err := wrapper.IamClient.CreateAccessKey(ctx, &iam.CreateAccessKeyInput{
  UserName: aws.String(userName)})
 if err != nil {
  log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
 userName, err)
 } else {
  key = result.AccessKey
 }
 return key, err
}



// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(ctx context.Context, userName string,
 keyId string) error {
 _, err := wrapper.IamClient.DeleteAccessKey(ctx, &iam.DeleteAccessKeyInput{
  AccessKeyId: aws.String(keyId),
  UserName:    aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
 }
 return err
}



// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(ctx context.Context, userName string)
 ([]types.AccessKeyMetadata, error) {
 var keys []types.AccessKeyMetadata
 result, err := wrapper.IamClient.ListAccessKeys(ctx, &iam.ListAccessKeysInput{
  UserName: aws.String(userName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
 err)
 } else {
  keys = result.AccessKeyMetadata
 }
 return keys, err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM user actions.

```
/*
```

```
  To run this Java V2 code example, set up your development environment,
 including your credentials.

  For information, see this documentation topic:

  https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html

  This example performs these operations:

  1. Creates a user that has no permissions.
  2. Creates a role and policy that grants Amazon S3 permissions.
  3. Creates a role.
  4. Grants the user permissions.
  5. Gets temporary credentials by assuming the role.  Creates an Amazon S3
 Service client object with the temporary credentials.
  6. Deletes the resources.
 */

public class IAMScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");
    public static final String PolicyDocument = "{" +
            "  \"Version\": \"2012-10-17\"," +
            "  \"Statement\": [" +
            "    {" +
            "        \"Effect\": \"Allow\"," +
            "        \"Action\": [" +
            "            \"s3:*\"" +
            "       ]," +
            "       \"Resource\": \"*\"" +
            "    }" +
            "   ]" +
            "}";

    public static String userArn;

    public static void main(String[] args) throws Exception {

        final String usage = """

                Usage:
                    <username> <policyName> <roleName> <roleSessionName>
 <bucketName>\s
```

```
                Where:
                    username - The name of the IAM user to create.\s
                    policyName - The name of the policy to create.\s
                    roleName - The name of the role to create.\s
                    roleSessionName - The name of the session required for the
assumeRole operation.\s
                    bucketName - The name of the Amazon S3 bucket from which
objects are read.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String userName = args[0];
        String policyName = args[1];
        String roleName = args[2];
        String roleSessionName = args[3];
        String bucketName = args[4];

        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the AWS IAM example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(" 1. Create the IAM user.");
        User createUser = createIAMUser(iam, userName);

        System.out.println(DASHES);
        userArn = createUser.arn();

        AccessKey myKey = createIAMAccessKey(iam, userName);
        String accessKey = myKey.accessKeyId();
        String secretKey = myKey.secretAccessKey();
        String assumeRolePolicyDocument = "{" +
                "\"Version\": \"2012-10-17\"," +
                "\"Statement\": [{" +
```

```
                "\"Effect\": \"Allow\"," +
                "\"Principal\": {" +
                " \"AWS\": \"" + userArn + "\"" +
                "}," +
                "\"Action\": \"sts:AssumeRole\"" +
                "}]" +
                "}";

        System.out.println(assumeRolePolicyDocument);
        System.out.println(userName + " was successfully created.");
        System.out.println(DASHES);
        System.out.println("2. Creates a policy.");
        String polArn = createIAMPolicy(iam, policyName);
        System.out.println("The policy " + polArn + " was successfully
created.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Creates a role.");
        TimeUnit.SECONDS.sleep(30);
        String roleArn = createIAMRole(iam, roleName, assumeRolePolicyDocument);
        System.out.println(roleArn + " was successfully created.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Grants the user permissions.");
        attachIAMRolePolicy(iam, roleName, polArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("*** Wait for 30 secs so the resource is available");
        TimeUnit.SECONDS.sleep(30);
        System.out.println("5. Gets temporary credentials by assuming the
role.");
        System.out.println("Perform an Amazon S3 Service operation using the
temporary credentials.");
        assumeRole(roleArn, roleSessionName, bucketName, accessKey, secretKey);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6 Getting ready to delete the AWS resources");
        deleteKey(iam, userName, accessKey);
        deleteRole(iam, roleName, polArn);
        deleteIAMUser(iam, userName);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("This IAM Scenario has successfully completed");
        System.out.println(DASHES);
    }

    public static AccessKey createIAMAccessKey(IamClient iam, String user) {
        try {
            CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
                    .userName(user)
                    .build();

            CreateAccessKeyResponse response = iam.createAccessKey(request);
            return response.accessKey();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static User createIAMUser(IamClient iam, String username) {
        try {
            // Create an IamWaiter object
            IamWaiter iamWaiter = iam.waiter();
            CreateUserRequest request = CreateUserRequest.builder()
                    .userName(username)
                    .build();

            // Wait until the user is created.
            CreateUserResponse response = iam.createUser(request);
            GetUserRequest userRequest = GetUserRequest.builder()
                    .userName(response.user().userName())
                    .build();

            WaiterResponse<GetUserResponse> waitUntilUserExists =
iamWaiter.waitUntilUserExists(userRequest);

waitUntilUserExists.matched().response().ifPresent(System.out::println);
            return response.user();

        } catch (IamException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static String createIAMRole(IamClient iam, String rolename, String
json) {

        try {
            CreateRoleRequest request = CreateRoleRequest.builder()
                    .roleName(rolename)
                    .assumeRolePolicyDocument(json)
                    .description("Created using the AWS SDK for Java")
                    .build();

            CreateRoleResponse response = iam.createRole(request);
            System.out.println("The ARN of the role is " +
response.role().arn());
            return response.role().arn();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createIAMPolicy(IamClient iam, String policyName) {
        try {
            // Create an IamWaiter object.
            IamWaiter iamWaiter = iam.waiter();
            CreatePolicyRequest request = CreatePolicyRequest.builder()
                    .policyName(policyName)
                    .policyDocument(PolicyDocument).build();

            CreatePolicyResponse response = iam.createPolicy(request);
            GetPolicyRequest polRequest = GetPolicyRequest.builder()
                    .policyArn(response.policy().arn())
                    .build();

            WaiterResponse<GetPolicyResponse> waitUntilPolicyExists =
iamWaiter.waitUntilPolicyExists(polRequest);
```

```
waitUntilPolicyExists.matched().response().ifPresent(System.out::println);
           return response.policy().arn();

       } catch (IamException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
       return "";
   }

   public static void attachIAMRolePolicy(IamClient iam, String roleName, String
policyArn) {
       try {
           ListAttachedRolePoliciesRequest request =
ListAttachedRolePoliciesRequest.builder()
                   .roleName(roleName)
                   .build();

           ListAttachedRolePoliciesResponse response =
iam.listAttachedRolePolicies(request);
           List<AttachedPolicy> attachedPolicies = response.attachedPolicies();
           String polArn;
           for (AttachedPolicy policy : attachedPolicies) {
               polArn = policy.policyArn();
               if (polArn.compareTo(policyArn) == 0) {
                   System.out.println(roleName + " policy is already attached to
this role.");
                   return;
               }
           }

           AttachRolePolicyRequest attachRequest =
AttachRolePolicyRequest.builder()
                   .roleName(roleName)
                   .policyArn(policyArn)
                   .build();

           iam.attachRolePolicy(attachRequest);
           System.out.println("Successfully attached policy " + policyArn + " to
role " + roleName);

       } catch (IamException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }

    // Invoke an Amazon S3 operation using the Assumed Role.
    public static void assumeRole(String roleArn, String roleSessionName, String
bucketName, String keyVal,
            String keySecret) {

        // Use the creds of the new IAM user that was created in this code
example.
        AwsBasicCredentials credentials = AwsBasicCredentials.create(keyVal,
keySecret);
        StsClient stsClient = StsClient.builder()
                .region(Region.US_EAST_1)

.credentialsProvider(StaticCredentialsProvider.create(credentials))
                .build();

        try {
            AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
                    .roleArn(roleArn)
                    .roleSessionName(roleSessionName)
                    .build();

            AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
            Credentials myCreds = roleResponse.credentials();
            String key = myCreds.accessKeyId();
            String secKey = myCreds.secretAccessKey();
            String secToken = myCreds.sessionToken();

            // List all objects in an Amazon S3 bucket using the temp creds
retrieved by
            // invoking assumeRole.
            Region region = Region.US_EAST_1;
            S3Client s3 = S3Client.builder()
                    .credentialsProvider(

StaticCredentialsProvider.create(AwsSessionCredentials.create(key, secKey,
secToken)))
                    .region(region)
                    .build();

            System.out.println("Created a S3Client using temp credentials.");
```

```
            System.out.println("Listing objects in " + bucketName);
            ListObjectsRequest listObjects = ListObjectsRequest.builder()
                    .bucket(bucketName)
                    .build();

            ListObjectsResponse res = s3.listObjects(listObjects);
            List<S3Object> objects = res.contents();
            for (S3Object myValue : objects) {
                System.out.println("The name of the key is " + myValue.key());
                System.out.println("The owner is " + myValue.owner());
            }

        } catch (StsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteRole(IamClient iam, String roleName, String polArn)
{

        try {
            // First the policy needs to be detached.
            DetachRolePolicyRequest rolePolicyRequest =
DetachRolePolicyRequest.builder()
                    .policyArn(polArn)
                    .roleName(roleName)
                    .build();

            iam.detachRolePolicy(rolePolicyRequest);

            // Delete the policy.
            DeletePolicyRequest request = DeletePolicyRequest.builder()
                    .policyArn(polArn)
                    .build();

            iam.deletePolicy(request);
            System.out.println("*** Successfully deleted " + polArn);

            // Delete the role.
            DeleteRoleRequest roleRequest = DeleteRoleRequest.builder()
                    .roleName(roleName)
                    .build();
```

```
            iam.deleteRole(roleRequest);
            System.out.println("*** Successfully deleted " + roleName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteKey(IamClient iam, String username, String
 accessKey) {
        try {
            DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()
                    .accessKeyId(accessKey)
                    .userName(username)
                    .build();

            iam.deleteAccessKey(request);
            System.out.println("Successfully deleted access key " + accessKey +
                    " from user " + username);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteIAMUser(IamClient iam, String userName) {
        try {
            DeleteUserRequest request = DeleteUserRequest.builder()
                    .userName(userName)
                    .build();

            iam.deleteUser(request);
            System.out.println("*** Successfully deleted " + userName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
import {
  CreateUserCommand,
  GetUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
```

```
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "iam_basic_test_username";
const policyName = "iam_basic_test_policy";
const roleName = "iam_basic_test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
```

```
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    }
    throw caught;
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service
 (AWS STS).
  // It's not best practice to use access keys. For more information, see
 https://aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
```

```
  },
 });

 // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
 // thrown while the user and access keys are still stabilizing.
 await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
   try {
     return await listBuckets(s3Client);
   } catch (err) {
     if (err instanceof Error && err.name === "InvalidAccessKeyId") {
       throw err;
     }
   }
 });

 // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
 // is thrown while the user and access keys are still stabilizing.
 const { Role } = await retry(
   {
     intervalInMs: 2000,
     maxRetries: 60,
   },
   () =>
     iamClient.send(
       new CreateRoleCommand({
         AssumeRolePolicyDocument: JSON.stringify({
           Version: "2012-10-17",
           Statement: [
             {
               Effect: "Allow",
               Principal: {
                 // Allow the previously created user to assume this role.
                 AWS: User.Arn,
               },
               Action: "sts:AssumeRole",
             },
           ],
         }),
         RoleName: roleName,
       }),
     ),
 );
```

```
if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3:ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  }),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
```

```
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 120 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

await iamClient.send(
  new DeletePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
  }),
);
```

```
  await iamClient.send(
    new DeleteRoleCommand({
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole

- CreateUser
- DeleteAccessKey
- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM user actions.

```
suspend fun main(args: Array<String>) {
    val usage = """
    Usage:
        <username> <policyName> <roleName> <roleSessionName> <fileLocation>
 <bucketName>

    Where:
        username - The name of the IAM user to create.
        policyName - The name of the policy to create.
        roleName - The name of the role to create.
        roleSessionName - The name of the session required for the assumeRole
 operation.
        fileLocation - The file location to the JSON required to create the role
 (see Readme).
        bucketName - The name of the Amazon S3 bucket from which objects are
 read.
    """
```

```
    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }

    val userName = args[0]
    val policyName = args[1]
    val roleName = args[2]
    val roleSessionName = args[3]
    val fileLocation = args[4]
    val bucketName = args[5]

    createUser(userName)
    println("$userName was successfully created.")

    val polArn = createPolicy(policyName)
    println("The policy $polArn was successfully created.")

    val roleArn = createRole(roleName, fileLocation)
    println("$roleArn was successfully created.")
    attachRolePolicy(roleName, polArn)

    println("*** Wait for 1 MIN so the resource is available.")
    delay(60000)
    assumeGivenRole(roleArn, roleSessionName, bucketName)

    println("*** Getting ready to delete the AWS resources.")
    deleteRole(roleName, polArn)
    deleteUser(userName)
    println("This IAM Scenario has successfully completed.")
}

suspend fun createUser(usernameVal: String?): String? {
    val request =
        CreateUserRequest {
            userName = usernameVal
        }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createUser(request)
        return response.user?.userName
    }
}
```

```
suspend fun createPolicy(policyNameVal: String?): String {
    val policyDocumentValue = """
    {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "s3:*"
                ],
                "Resource": "*"
            }
        ]
    }
    """.trimIndent()

    val request =
        CreatePolicyRequest {
            policyName = policyNameVal
            policyDocument = policyDocumentValue
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createPolicy(request)
        return response.policy?.arn.toString()
    }
}

suspend fun createRole(
    rolenameVal: String?,
    fileLocation: String?,
): String? {
    val jsonObject = fileLocation?.let { readJsonSimpleDemo(it) } as JSONObject

    val request =
        CreateRoleRequest {
            roleName = rolenameVal
            assumeRolePolicyDocument = jsonObject.toJSONString()
            description = "Created using the AWS SDK for Kotlin"
        }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createRole(request)
        return response.role?.arn
```

```
    }
}

suspend fun attachRolePolicy(
    roleNameVal: String,
    policyArnVal: String,
) {
    val request =
        ListAttachedRolePoliciesRequest {
            roleName = roleNameVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAttachedRolePolicies(request)
        val attachedPolicies = response.attachedPolicies

        // Ensure that the policy is not attached to this role.
        val checkStatus: Int
        if (attachedPolicies != null) {
            checkStatus = checkMyList(attachedPolicies, policyArnVal)
            if (checkStatus == -1) {
                return
            }
        }

        val policyRequest =
            AttachRolePolicyRequest {
                roleName = roleNameVal
                policyArn = policyArnVal
            }
        iamClient.attachRolePolicy(policyRequest)
        println("Successfully attached policy $policyArnVal to role
 $roleNameVal")
    }
}

fun checkMyList(
    attachedPolicies: List<AttachedPolicy>,
    policyArnVal: String,
): Int {
    for (policy in attachedPolicies) {
        val polArn = policy.policyArn.toString()

        if (polArn.compareTo(policyArnVal) == 0) {
```

```
            println("The policy is already attached to this role.")
            return -1
        }
    }
    return 0
}

suspend fun assumeGivenRole(
    roleArnVal: String?,
    roleSessionNameVal: String?,
    bucketName: String,
) {
    val stsClient = StsClient.fromEnvironment { region = "us-east-1" }
    val roleRequest =
        AssumeRoleRequest {
            roleArn = roleArnVal
            roleSessionName = roleSessionNameVal
        }

    val roleResponse = stsClient.assumeRole(roleRequest)
    val myCreds = roleResponse.credentials
    val key = myCreds?.accessKeyId
    val secKey = myCreds?.secretAccessKey
    val secToken = myCreds?.sessionToken

    val staticCredentials = StaticCredentialsProvider {
        accessKeyId = key
        secretAccessKey = secKey
        sessionToken = secToken
    }

    // List all objects in an Amazon S3 bucket using the temp creds.
    val s3 = S3Client.fromEnvironment {
        region = "us-east-1"
        credentialsProvider = staticCredentials
    }

    println("Created a S3Client using temp credentials.")
    println("Listing objects in $bucketName")

    val listObjects =
        ListObjectsRequest {
            bucket = bucketName
        }
```

```
    val response = s3.listObjects(listObjects)
    response.contents?.forEach { myObject ->
        println("The name of the key is ${myObject.key}")
        println("The owner is ${myObject.owner}")
    }
}

suspend fun deleteRole(
    roleNameVal: String,
    polArn: String,
) {
    val iam = IamClient.fromEnvironment { region = "AWS_GLOBAL" }

    // First the policy needs to be detached.
    val rolePolicyRequest =
        DetachRolePolicyRequest {
            policyArn = polArn
            roleName = roleNameVal
        }

    iam.detachRolePolicy(rolePolicyRequest)

    // Delete the policy.
    val request =
        DeletePolicyRequest {
            policyArn = polArn
        }

    iam.deletePolicy(request)
    println("*** Successfully deleted $polArn")

    // Delete the role.
    val roleRequest =
        DeleteRoleRequest {
            roleName = roleNameVal
        }

    iam.deleteRole(roleRequest)
    println("*** Successfully deleted $roleNameVal")
}

suspend fun deleteUser(userNameVal: String) {
    val iam = IamClient.fromEnvironment { region = "AWS_GLOBAL" }
```

```
    val request =
        DeleteUserRequest {
            userName = userNameVal
        }

    iam.deleteUser(request)
    println("*** Successfully deleted $userNameVal")
}

@Throws(java.lang.Exception::class)
fun readJsonSimpleDemo(filename: String): Any? {
    val reader = FileReader(filename)
    val jsonParser = JSONParser()
    return jsonParser.parse(reader)
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace Iam\Basics;

require 'vendor/autoload.php';

use Aws\Credentials\Credentials;
use Aws\S3\Exception\S3Exception;
use Aws\S3\S3Client;
use Aws\Sts\StsClient;
use Iam\IAMService;

echo("\n");
echo("--------------------------------------\n");
print("Welcome to the IAM getting started demo using PHP!\n");
echo("--------------------------------------\n");

$uuid = uniqid();
$service = new IAMService();

$user = $service->createUser("iam_demo_user_$uuid");
echo "Created user with the arn: {$user['Arn']}\n";

$key = $service->createAccessKey($user['UserName']);
$assumeRolePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
 $assumeRolePolicyDocument);
```

```
echo "Created role: {$assumeRoleRole['RoleName']}\n";

$listAllBucketsPolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
 $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

$service->attachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);

$inlinePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"sts:AssumeRole\",
                    \"Resource\": \"{$assumeRoleRole['Arn']}\"}]
}";
$inlinePolicy = $service->createUserPolicy("iam_demo_inline_policy_$uuid",
 $inlinePolicyDocument, $user['UserName']);
//First, fail to list the buckets with the user
$credentials = new Credentials($key['AccessKeyId'], $key['SecretAccessKey']);
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $credentials]);
try {
    $s3Client->listBuckets([
    ]);
    echo "this should not run";
} catch (S3Exception $exception) {
    echo "successfully failed!\n";
}

$stsClient = new StsClient(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $credentials]);
sleep(10);
$assumedRole = $stsClient->assumeRole([
    'RoleArn' => $assumeRoleRole['Arn'],
    'RoleSessionName' => "DemoAssumeRoleSession_$uuid",
]);
```

```
$assumedCredentials = [
    'key' => $assumedRole['Credentials']['AccessKeyId'],
    'secret' => $assumedRole['Credentials']['SecretAccessKey'],
    'token' => $assumedRole['Credentials']['SessionToken'],
];
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $assumedCredentials]);
try {
    $s3Client->listBuckets([]);
    echo "this should now run!\n";
} catch (S3Exception $exception) {
    echo "this should now not fail\n";
}

$service->detachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);
$deletePolicy = $service->deletePolicy($listAllBucketsPolicy['Arn']);
echo "Delete policy: {$listAllBucketsPolicy['PolicyName']}\n";
$deletedRole = $service->deleteRole($assumeRoleRole['Arn']);
echo "Deleted role: {$assumeRoleRole['RoleName']}\n";
$deletedKey = $service->deleteAccessKey($key['AccessKeyId'], $user['UserName']);
$deletedUser = $service->deleteUser($user['UserName']);
echo "Delete user: {$user['UserName']}\n";
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy

- PutUserPolicy

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
import json
import sys
import time
from uuid import uuid4

import boto3
from botocore.exceptions import ClientError


def progress_bar(seconds):
    """Shows a simple progress bar in the command window."""
    for _ in range(seconds):
        time.sleep(1)
        print(".", end="")
        sys.stdout.flush()
    print()


def setup(iam_resource):
    """
    Creates a new user with no permissions.
    Creates an access key pair for the user.
    Creates a role with a policy that lets the user assume the role.
    Creates a policy that allows listing Amazon S3 buckets.
    Attaches the policy to the role.
```

```
    Creates an inline policy for the user that lets the user assume the role.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM)
resource
                         that has permissions to create users, roles, and
policies
                         in the account.
    :return: The newly created user, user key, and role.
    """
    try:
        user = iam_resource.create_user(UserName=f"demo-user-{uuid4()}")
        print(f"Created user {user.name}.")
    except ClientError as error:
        print(
            f"Couldn't create a user for the demo. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    try:
        user_key = user.create_access_key_pair()
        print(f"Created access key pair for user.")
    except ClientError as error:
        print(
            f"Couldn't create access keys for user {user.name}. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    print(f"Wait for user to be ready.", end="")
    progress_bar(10)

    try:
        role = iam_resource.create_role(
            RoleName=f"demo-role-{uuid4()}",
            AssumeRolePolicyDocument=json.dumps(
                {
                    "Version":"2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {"AWS": user.arn},
                            "Action": "sts:AssumeRole",
                        }
```

```
                    ],
                }
            ),
        )
        print(f"Created role {role.name}.")
    except ClientError as error:
        print(
            f"Couldn't create a role for the demo. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    try:
        policy = iam_resource.create_policy(
            PolicyName=f"demo-policy-{uuid4()}",
            PolicyDocument=json.dumps(
                {
                    "Version":"2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": "s3:ListAllMyBuckets",
                            "Resource": "arn:aws:s3:::*",
                        }
                    ],
                }
            ),
        )
        role.attach_policy(PolicyArn=policy.arn)
        print(f"Created policy {policy.policy_name} and attached it to the
role.")
    except ClientError as error:
        print(
            f"Couldn't create a policy and attach it to role {role.name}. Here's
why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    try:
        user.create_policy(
            PolicyName=f"demo-user-policy-{uuid4()}",
            PolicyDocument=json.dumps(
                {
```

```
                    "Version":"2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": "sts:AssumeRole",
                            "Resource": role.arn,
                        }
                    ],
                }
            ),
        )
        print(
            f"Created an inline policy for {user.name} that lets the user assume
 "
            f"the role."
        )
    except ClientError as error:
        print(
            f"Couldn't create an inline policy for user {user.name}. Here's why:
 "
            f"{error.response['Error']['Message']}"
        )
        raise

    print("Give AWS time to propagate these new resources and connections.",
 end="")
    progress_bar(10)

    return user, user_key, role


def show_access_denied_without_role(user_key):
    """
    Shows that listing buckets without first assuming the role is not allowed.

    :param user_key: The key of the user created during setup. This user does not
                     have permission to list buckets in the account.
    """
    print(f"Try to list buckets without first assuming the role.")
    s3_denied_resource = boto3.resource(
        "s3", aws_access_key_id=user_key.id,
 aws_secret_access_key=user_key.secret
    )
    try:
```

```
        for bucket in s3_denied_resource.buckets.all():
            print(bucket.name)
        raise RuntimeError("Expected to get AccessDenied error when listing
 buckets!")
    except ClientError as error:
        if error.response["Error"]["Code"] == "AccessDenied":
            print("Attempt to list buckets with no permissions: AccessDenied.")
        else:
            raise


def list_buckets_from_assumed_role(user_key, assume_role_arn, session_name):
    """
    Assumes a role that grants permission to list the Amazon S3 buckets in the
 account.
    Uses the temporary credentials from the role to list the buckets that are
 owned
    by the assumed role's account.

    :param user_key: The access key of a user that has permission to assume the
 role.
    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                            grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    """
    sts_client = boto3.client(
        "sts", aws_access_key_id=user_key.id,
 aws_secret_access_key=user_key.secret
    )
    try:
        response = sts_client.assume_role(
            RoleArn=assume_role_arn, RoleSessionName=session_name
        )
        temp_credentials = response["Credentials"]
        print(f"Assumed role {assume_role_arn} and got temporary credentials.")
    except ClientError as error:
        print(
            f"Couldn't assume role {assume_role_arn}. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    # Create an S3 resource that can access the account with the temporary
 credentials.
```

```
    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )
    print(f"Listing buckets for the assumed role's account:")
    try:
        for bucket in s3_resource.buckets.all():
            print(bucket.name)
    except ClientError as error:
        print(
            f"Couldn't list buckets for the account. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise




def teardown(user, role):
    """
    Removes all resources created during setup.

    :param user: The demo user.
    :param role: The demo role.
    """
    try:
        for attached in role.attached_policies.all():
            policy_name = attached.policy_name
            role.detach_policy(PolicyArn=attached.arn)
            attached.delete()
            print(f"Detached and deleted {policy_name}.")
        role.delete()
        print(f"Deleted {role.name}.")
    except ClientError as error:
        print(
            "Couldn't detach policy, delete policy, or delete role. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    try:
        for user_pol in user.policies.all():
```

```
            user_pol.delete()
            print("Deleted inline user policy.")
        for key in user.access_keys.all():
            key.delete()
            print("Deleted user's access key.")
        user.delete()
        print(f"Deleted {user.name}.")
    except ClientError as error:
        print(
            "Couldn't delete user policy or delete user. Here's why: "
            f"{error.response['Error']['Message']}"
        )


def usage_demo():
    """Drives the demonstration."""
    print("-" * 88)
    print(f"Welcome to the IAM create user and assume role demo.")
    print("-" * 88)
    iam_resource = boto3.resource("iam")
    user = None
    role = None
    try:
        user, user_key, role = setup(iam_resource)
        print(f"Created {user.name} and {role.name}.")
        show_access_denied_without_role(user_key)
        list_buckets_from_assumed_role(user_key, role.arn,
 "AssumeRoleDemoSession")
    except Exception:
        print("Something went wrong!")
    finally:
        if user is not None and role is not None:
            teardown(user, role)
        print("Thanks for watching!")


if __name__ == "__main__":
    usage_demo()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - AttachRolePolicy
  - CreateAccessKey

- CreatePolicy
- CreateRole
- CreateUser
- DeleteAccessKey
- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
# Wraps the scenario actions.
class ScenarioCreateUserAssumeRole
  attr_reader :iam_client

  # @param [Aws::IAM::Client] iam_client: The AWS IAM client.
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
  end

  # Waits for the specified number of seconds.
  #
```

```
  # @param duration [Integer] The number of seconds to wait.
  def wait(duration)
    puts('Give AWS time to propagate resources...')
    sleep(duration)
  end

  # Creates a user.
  #
  # @param user_name [String] The name to give the user.
  # @return [Aws::IAM::User] The newly created user.
  def create_user(user_name)
    user = @iam_client.create_user(user_name: user_name).user
    @logger.info("Created demo user named #{user.user_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info('Tried and failed to create demo user.')
    @logger.info("\t#{e.code}: #{e.message}")
    @logger.info("\nCan't continue the demo without a user!")
    raise
  else
    user
  end

  # Creates an access key for a user.
  #
  # @param user [Aws::IAM::User] The user that owns the key.
  # @return [Aws::IAM::AccessKeyPair] The newly created access key.
  def create_access_key_pair(user)
    user_key = @iam_client.create_access_key(user_name:
user.user_name).access_key
    @logger.info("Created accesskey pair for user #{user.user_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info("Couldn't create access keys for user #{user.user_name}.")
    @logger.info("\t#{e.code}: #{e.message}")
    raise
  else
    user_key
  end

  # Creates a role that can be assumed by a user.
  #
  # @param role_name [String] The name to give the role.
  # @param user [Aws::IAM::User] The user who is granted permission to assume the
role.
  # @return [Aws::IAM::Role] The newly created role.
```

```
  def create_role(role_name, user)
    trust_policy = {
      Version: '2012-10-17',
      Statement: [{
        Effect: 'Allow',
        Principal: { 'AWS': user.arn },
        Action: 'sts:AssumeRole'
      }]
    }.to_json
    role = @iam_client.create_role(
      role_name: role_name,
      assume_role_policy_document: trust_policy
    ).role
    @logger.info("Created role #{role.role_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info("Couldn't create a role for the demo. Here's why: ")
    @logger.info("\t#{e.code}: #{e.message}")
    raise
  else
    role
  end

  # Creates a policy that grants permission to list S3 buckets in the account,
and
  # then attaches the policy to a role.
  #
  # @param policy_name [String] The name to give the policy.
  # @param role [Aws::IAM::Role] The role that the policy is attached to.
  # @return [Aws::IAM::Policy] The newly created policy.
  def create_and_attach_role_policy(policy_name, role)
    policy_document = {
      Version: '2012-10-17',
      Statement: [{
        Effect: 'Allow',
        Action: 's3:ListAllMyBuckets',
        Resource: 'arn:aws:s3:::*'
      }]
    }.to_json
    policy = @iam_client.create_policy(
      policy_name: policy_name,
      policy_document: policy_document
    ).policy
    @iam_client.attach_role_policy(
      role_name: role.role_name,
```

```
      policy_arn: policy.arn
    )
    @logger.info("Created policy #{policy.policy_name} and attached it to role
#{role.role_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info("Couldn't create a policy and attach it to role
#{role.role_name}. Here's why: ")
    @logger.info("\t#{e.code}: #{e.message}")
    raise
  end

  # Creates an inline policy for a user that lets the user assume a role.
  #
  # @param policy_name [String] The name to give the policy.
  # @param user [Aws::IAM::User] The user that owns the policy.
  # @param role [Aws::IAM::Role] The role that can be assumed.
  # @return [Aws::IAM::UserPolicy] The newly created policy.
  def create_user_policy(policy_name, user, role)
    policy_document = {
      Version: '2012-10-17',
      Statement: [{
        Effect: 'Allow',
        Action: 'sts:AssumeRole',
        Resource: role.arn
      }]
    }.to_json
    @iam_client.put_user_policy(
      user_name: user.user_name,
      policy_name: policy_name,
      policy_document: policy_document
    )
    puts("Created an inline policy for #{user.user_name} that lets the user
assume role #{role.role_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info("Couldn't create an inline policy for user #{user.user_name}.
Here's why: ")
    @logger.info("\t#{e.code}: #{e.message}")
    raise
  end

  # Creates an Amazon S3 resource with specified credentials. This is separated
into a
  # factory function so that it can be mocked for unit testing.
  #
```

```
 # @param credentials [Aws::Credentials] The credentials used by the Amazon S3
resource.
 def create_s3_resource(credentials)
   Aws::S3::Resource.new(client: Aws::S3::Client.new(credentials: credentials))
 end

 # Lists the S3 buckets for the account, using the specified Amazon S3 resource.
 # Because the resource uses credentials with limited access, it may not be able
to
 # list the S3 buckets.
 #
 # @param s3_resource [Aws::S3::Resource] An Amazon S3 resource.
 def list_buckets(s3_resource)
   count = 10
   s3_resource.buckets.each do |bucket|
     @logger.info "\t#{bucket.name}"
     count -= 1
     break if count.zero?
   end
 rescue Aws::Errors::ServiceError => e
   if e.code == 'AccessDenied'
     puts('Attempt to list buckets with no permissions: AccessDenied.')
   else
     @logger.info("Couldn't list buckets for the account. Here's why: ")
     @logger.info("\t#{e.code}: #{e.message}")
     raise
   end
 end

 # Creates an AWS Security Token Service (AWS STS) client with specified
credentials.
 # This is separated into a factory function so that it can be mocked for unit
testing.
 #
 # @param key_id [String] The ID of the access key used by the STS client.
 # @param key_secret [String] The secret part of the access key used by the STS
client.
 def create_sts_client(key_id, key_secret)
   Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)
 end

 # Gets temporary credentials that can be used to assume a role.
 #
```

```
 # @param role_arn [String] The ARN of the role that is assumed when these
credentials
 #                          are used.
 # @param sts_client [AWS::STS::Client] An AWS STS client.
 # @return [Aws::AssumeRoleCredentials] The credentials that can be used to
assume the role.
 def assume_role(role_arn, sts_client)
   credentials = Aws::AssumeRoleCredentials.new(
     client: sts_client,
     role_arn: role_arn,
     role_session_name: 'create-use-assume-role-scenario'
   )
   @logger.info("Assumed role '#{role_arn}', got temporary credentials.")
   credentials
 end

 # Deletes a role. If the role has policies attached, they are detached and
 # deleted before the role is deleted.
 #
 # @param role_name [String] The name of the role to delete.
 def delete_role(role_name)
   @iam_client.list_attached_role_policies(role_name:
role_name).attached_policies.each do |policy|
     @iam_client.detach_role_policy(role_name: role_name, policy_arn:
policy.policy_arn)
     @iam_client.delete_policy(policy_arn: policy.policy_arn)
     @logger.info("Detached and deleted policy #{policy.policy_name}.")
   end
   @iam_client.delete_role({ role_name: role_name })
   @logger.info("Role deleted: #{role_name}.")
 rescue Aws::Errors::ServiceError => e
   @logger.info("Couldn't detach policies and delete role #{role.name}. Here's
why:")
   @logger.info("\t#{e.code}: #{e.message}")
   raise
 end

 # Deletes a user. If the user has inline policies or access keys, they are
deleted
 # before the user is deleted.
 #
 # @param user [Aws::IAM::User] The user to delete.
 def delete_user(user_name)
   user = @iam_client.list_access_keys(user_name: user_name).access_key_metadata
```

```
    user.each do |key|
      @iam_client.delete_access_key({ access_key_id: key.access_key_id,
 user_name: user_name })
      @logger.info("Deleted access key #{key.access_key_id} for user
 '#{user_name}'.")
    end

    @iam_client.delete_user(user_name: user_name)
    @logger.info("Deleted user '#{user_name}'.")
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting user '#{user_name}': #{e.message}")
  end
end

# Runs the IAM create a user and assume a role scenario.
def run_scenario(scenario)
  puts('-' * 88)
  puts('Welcome to the IAM create a user and assume a role demo!')
  puts('-' * 88)
  user = scenario.create_user("doc-example-user-#{Random.uuid}")
  user_key = scenario.create_access_key_pair(user)
  scenario.wait(10)
  role = scenario.create_role("doc-example-role-#{Random.uuid}", user)
  scenario.create_and_attach_role_policy("doc-example-role-policy-
#{Random.uuid}", role)
  scenario.create_user_policy("doc-example-user-policy-#{Random.uuid}", user,
 role)
  scenario.wait(10)
  puts('Try to list buckets with credentials for a user who has no permissions.')
  puts('Expect AccessDenied from this call.')
  scenario.list_buckets(
    scenario.create_s3_resource(Aws::Credentials.new(user_key.access_key_id,
 user_key.secret_access_key))
  )
  puts('Now, assume the role that grants permission.')
  temp_credentials = scenario.assume_role(
    role.arn, scenario.create_sts_client(user_key.access_key_id,
 user_key.secret_access_key)
  )
  puts('Here are your buckets:')
  scenario.list_buckets(scenario.create_s3_resource(temp_credentials))
  puts("Deleting role '#{role.role_name}' and attached policies.")
  scenario.delete_role(role.role_name)
  puts("Deleting user '#{user.user_name}', policies, and keys.")
```

```
  scenario.delete_user(user.user_name)
  puts('Thanks for watching!')
  puts('-' * 88)
rescue Aws::Errors::ServiceError => e
  puts('Something went wrong with the demo.')
  puts("\t#{e.code}: #{e.message}")
end

run_scenario(ScenarioCreateUserAssumeRole.new(Aws::IAM::Client.new)) if
 $PROGRAM_NAME == __FILE__
```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
use aws_config::meta::region::RegionProviderChain;
use aws_sdk_iam::Error as iamError;
use aws_sdk_iam::{config::Credentials as iamCredentials, config::Region, Client
 as iamClient};
use aws_sdk_s3::Client as s3Client;
use aws_sdk_sts::Client as stsClient;
use tokio::time::{sleep, Duration};
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), iamError> {
    let (client, uuid, list_all_buckets_policy_document, inline_policy_document)
 =
        initialize_variables().await;

    if let Err(e) = run_iam_operations(
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
    .await
    {
        println!("{:?}", e);
    };

    Ok(())
}

async fn initialize_variables() -> (iamClient, String, String, String) {
    let region_provider = RegionProviderChain::first_try(Region::new("us-
west-2"));

    let shared_config =
 aws_config::from_env().region(region_provider).load().await;
    let client = iamClient::new(&shared_config);
    let uuid = Uuid::new_v4().to_string();

    let list_all_buckets_policy_document = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
```

```
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
    }"
    .to_string();
    let inline_policy_document = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"sts:AssumeRole\",
                    \"Resource\": \"{}\"}]
    }"
    .to_string();

    (
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
}

async fn run_iam_operations(
    client: iamClient,
    uuid: String,
    list_all_buckets_policy_document: String,
    inline_policy_document: String,
) -> Result<(), iamError> {
    let user = iam_service::create_user(&client, &format!("{}{}",
 "iam_demo_user_", uuid)).await?;
    println!("Created the user with the name: {}", user.user_name());
    let key = iam_service::create_access_key(&client, user.user_name()).await?;

    let assume_role_policy_document = "{
        \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }"
    .to_string()
    .replace("{}", user.arn());

    let assume_role_role = iam_service::create_role(
```

```
        &client,
        &format!("{}{}", "iam_demo_role_", uuid),
        &assume_role_policy_document,
    )
    .await?;
    println!("Created the role with the ARN: {}", assume_role_role.arn());

    let list_all_buckets_policy = iam_service::create_policy(
        &client,
        &format!("{}{}", "iam_demo_policy_", uuid),
        &list_all_buckets_policy_document,
    )
    .await?;
    println!(
        "Created policy: {}",
        list_all_buckets_policy.policy_name.as_ref().unwrap()
    );

    let attach_role_policy_result =
        iam_service::attach_role_policy(&client, &assume_role_role,
&list_all_buckets_policy)
            .await?;
    println!(
        "Attached the policy to the role: {:?}",
        attach_role_policy_result
    );

    let inline_policy_name = format!("{}{}", "iam_demo_inline_policy_", uuid);
    let inline_policy_document = inline_policy_document.replace("{}",
assume_role_role.arn());
    iam_service::create_user_policy(&client, &user, &inline_policy_name,
&inline_policy_document)
        .await?;
    println!("Created inline policy.");

    //First, fail to list the buckets with the user.
    let creds = iamCredentials::from_keys(key.access_key_id(),
key.secret_access_key(), None);
    let fail_config = aws_config::from_env()
        .credentials_provider(creds.clone())
        .load()
        .await;
    println!("Fail config: {:?}", fail_config);
    let fail_client: s3Client = s3Client::new(&fail_config);
```

```
    match fail_client.list_buckets().send().await {
        Ok(e) => {
            println!("This should not run. {:?}", e);
        }
        Err(e) => {
            println!("Successfully failed with error: {:?}", e)
        }
    }

    let sts_config = aws_config::from_env()
        .credentials_provider(creds.clone())
        .load()
        .await;
    let sts_client: stsClient = stsClient::new(&sts_config);
    sleep(Duration::from_secs(10)).await;
    let assumed_role = sts_client
        .assume_role()
        .role_arn(assume_role_role.arn())
        .role_session_name(format!("iam_demo_assumerole_session_{uuid}"))
        .send()
        .await;
    println!("Assumed role: {:?}", assumed_role);
    sleep(Duration::from_secs(10)).await;

    let assumed_credentials = iamCredentials::from_keys(
        assumed_role
            .as_ref()
            .unwrap()
            .credentials
            .as_ref()
            .unwrap()
            .access_key_id(),
        assumed_role
            .as_ref()
            .unwrap()
            .credentials
            .as_ref()
            .unwrap()
            .secret_access_key(),
        Some(
            assumed_role
                .as_ref()
                .unwrap()
                .credentials
```

```
                .as_ref()
                .unwrap()
                .session_token
                .clone(),
        ),
    );

    let succeed_config = aws_config::from_env()
        .credentials_provider(assumed_credentials)
        .load()
        .await;
    println!("succeed config: {:?}", succeed_config);
    let succeed_client: s3Client = s3Client::new(&succeed_config);
    sleep(Duration::from_secs(10)).await;
    match succeed_client.list_buckets().send().await {
        Ok(_) => {
            println!("This should now run successfully.")
        }
        Err(e) => {
            println!("This should not run. {:?}", e);
            panic!()
        }
    }

    //Clean up.
    iam_service::detach_role_policy(
        &client,
        assume_role_role.role_name(),
        list_all_buckets_policy.arn().unwrap_or_default(),
    )
    .await?;
    iam_service::delete_policy(&client, list_all_buckets_policy).await?;
    iam_service::delete_role(&client, &assume_role_role).await?;
    println!("Deleted role {}", assume_role_role.role_name());
    iam_service::delete_access_key(&client, &user, &key).await?;
    println!("Deleted key for {}", key.user_name());
    iam_service::delete_user_policy(&client, &user, &inline_policy_name).await?;
    println!("Deleted inline user policy: {}", inline_policy_name);
    iam_service::delete_user(&client, &user).await?;
    println!("Deleted user {}", user.user_name());

    Ok(())
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Use an IAM role to grant permissions to applications running on Amazon EC2 instances

Applications that run on an Amazon EC2 instance must include AWS credentials in the AWS API requests. You could have your developers store AWS credentials directly within the Amazon EC2 instance and allow applications in that instance to use those credentials. But developers would then have to manage the credentials and ensure that they securely pass the credentials to each instance and update each Amazon EC2 instance when it's time to update the credentials. That's a lot of additional work.

Instead, you can and should use an IAM role to manage *temporary* credentials for applications that run on an Amazon EC2 instance. When you use a role, you don't have to distribute long-term credentials (such as sign-in credentials or access keys) to an Amazon EC2 instance. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources. When you launch an Amazon EC2 instance, you specify an IAM role to associate with the instance. Applications that run on the instance can then use the role-supplied temporary credentials to sign API requests.

Using roles to grant permissions to applications that run on Amazon EC2 instances requires a bit of extra configuration. An application running on an Amazon EC2 instance is abstracted from AWS by the virtualized operating system. Because of this extra separation, you need an additional step to assign an AWS role and its associated permissions to an Amazon EC2 instance and make them available to its applications. This extra step is the creation of an *instance profile* attached to the instance. The instance profile contains the role and can provide the role's temporary credentials to an application that runs on the instance. Those temporary credentials can then be used in the application's API calls to access resources and to limit access to only those resources that the role specifies.

> **Note**
>
> Only one role can be assigned to an Amazon EC2 instance at a time, and all applications on the instance share the same role and permissions. When you leverage Amazon ECS to manage your Amazon EC2 instances, you can assign roles to Amazon ECS tasks that can be distinguished from the role of the Amazon EC2 instance that it's running on. Assigning each task a role aligns with the principle of least privileged access and allows for greater granular control over actions and resources.
> For more information, see Using IAM roles with Amazon ECS tasks in the *Amazon Elastic Container Service Best Practices Guide*.

Using roles in this way has several benefits. Because role credentials are temporary and updated automatically, you don't have to manage credentials, and you don't have to worry about long-term security risks. In addition, if you use a single role for multiple instances, you can make a change to that one role and the change propagates automatically to all the instances.

> **Note**
>
> Although a role is usually assigned to an Amazon EC2 instance when you launch it, a role can also be attached to an Amazon EC2 instance currently running. To learn how to attach a role to a running instance, see IAM Roles for Amazon EC2.

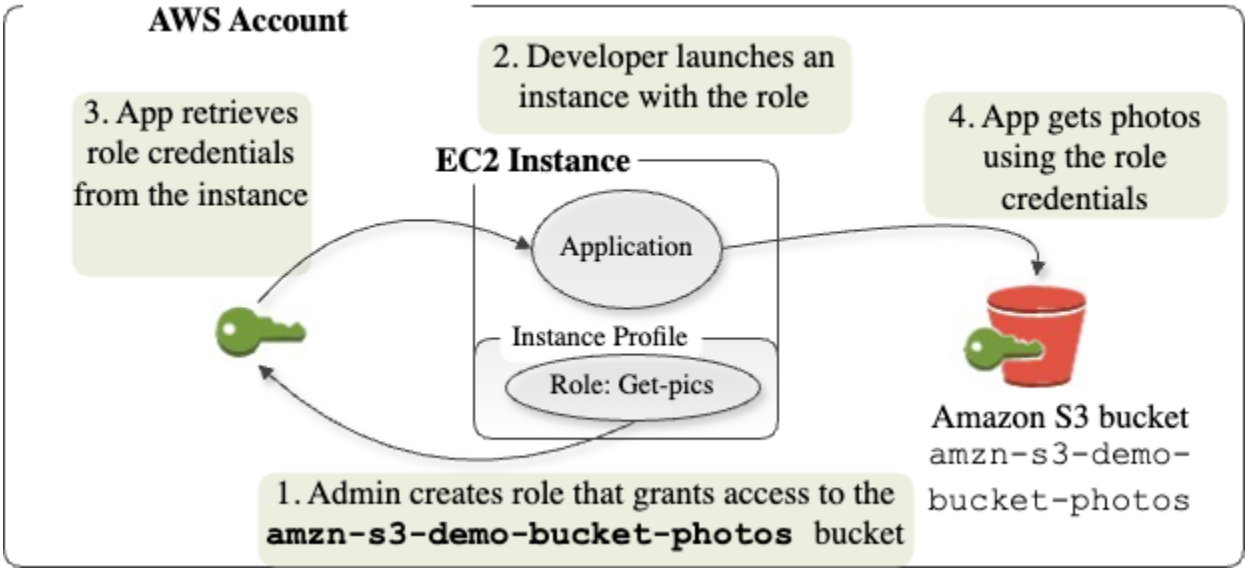
Topics

- How do roles for Amazon EC2 instances work?
- Permissions required for using roles with Amazon EC2

- How do I get started?
- Related information

How do roles for Amazon EC2 instances work?

In the following figure, a developer runs an application on an Amazon EC2 instance that requires access to the S3 bucket named `amzn-s3-demo-bucket-photos`. An administrator creates the `Get-pics` service role and attaches the role to the Amazon EC2 instance. The role includes a permissions policy that grants read-only access to the specified S3 bucket. It also includes a trust policy that allows the Amazon EC2 instance to assume the role and retrieve the temporary credentials. When the application runs on the instance, it can use the role's temporary credentials to access the photos bucket. The administrator doesn't have to grant the developer permission to access the photos bucket, and the developer never has to share or manage credentials.



1. The administrator uses IAM to create the **`Get-pics`** role. In the role's trust policy, the administrator specifies that only Amazon EC2 instances can assume the role. In the role's permission policy, the administrator specifies read-only permissions for the `amzn-s3-demo-bucket-photos` bucket.
2. A developer launches an Amazon EC2 instance and assigns the `Get-pics` role to that instance.

> **Note**
>
> If you use the IAM console, the instance profile is managed for you and is mostly transparent to you. However, if you use the AWS CLI or API to create and manage the

role and Amazon EC2 instance, then you must create the instance profile and assign the role to it as separate steps. Then, when you launch the instance, you must specify the instance profile name instead of the role name.

3. When the application runs, it obtains temporary security credentials from Amazon EC2 instance metadata, as described in Retrieving Security Credentials from Instance Metadata. These are temporary security credentials that represent the role and are valid for a limited period of time.

   With some AWS SDKs, the developer can use a provider that manages the temporary security credentials transparently. (The documentation for individual AWS SDKs describes the features supported by that SDK for managing credentials.)

   Alternatively, the application can get the temporary credentials directly from the instance metadata of the Amazon EC2 instance. Credentials and related values are available from the `iam/security-credentials/role-name` category (in this case, `iam/security-credentials/Get-pics`) of the metadata. If the application gets the credentials from the instance metadata, it can cache the credentials.

4. Using the retrieved temporary credentials, the application accesses the photo bucket. Because of the policy attached to the **`Get-pics`** role, the application has read-only permissions.

   The temporary security credentials available on the instance automatically update before they expire so that a valid set is always available. The application just needs to make sure that it gets a new set of credentials from the instance metadata before the current ones expire. It is possible to use the AWS SDK to manage credentials so the application does not need to include additional logic to refresh the credentials. For example, instantiating clients with Instance Profile Credential Providers. However, if the application gets temporary security credentials from the instance metadata and has cached them, it should get a refreshed set of credentials every hour, or at least 15 minutes before the current set expires. The expiration time is included in the information returned in the `iam/security-credentials/role-name` category.

Permissions required for using roles with Amazon EC2

To launch an instance with a role, the developer must have permission to launch Amazon EC2 instances and permission to pass IAM roles.

The following sample policy allows users to use the AWS Management Console to launch an instance with a role. The policy includes wildcards (*) to allow a user to pass any role and to

perform the listed Amazon EC2 actions. The `ListInstanceProfiles` action allows users to view all of the roles available in the AWS account.

Example policy that grants a user permission to use the Amazon EC2 console to launch an instance with any role

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "IamPassRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "ec2.amazonaws.com"
                }
            }
        },
        {
            "Sid": "ListEc2AndListInstanceProfiles",
            "Effect": "Allow",
            "Action": [
                "iam:ListInstanceProfiles",
                "ec2:Describe*",
                "ec2:Search*",
                "ec2:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

Restricting which roles can be passed to Amazon EC2 instances (using PassRole)

You can use the `PassRole` permission to restrict which role a user can pass to an Amazon EC2 instance when the user launches the instance. This helps prevent the user from running

applications that have more permissions than the user has been granted—that is, from being able to obtain elevated privileges. For example, imagine that user Alice has permissions only to launch Amazon EC2 instances and to work with Amazon S3 buckets, but the role she passes to an Amazon EC2 instance has permissions to work with IAM and Amazon DynamoDB. In that case, Alice might be able to launch the instance, log into it, get temporary security credentials, and then perform IAM or DynamoDB actions that she's not authorized for.

To restrict which roles a user can pass to an Amazon EC2 instance, you create a policy that allows the `PassRole` action. You then attach the policy to the user (or to an IAM group that the user belongs to) who will launch Amazon EC2 instances. In the `Resource` element of the policy, you list the role or roles that the user is allowed to pass to Amazon EC2 instances. When the user launches an instance and associates a role with it, Amazon EC2 checks whether the user is allowed to pass that role. Of course, you should also ensure that the role that the user can pass does not include more permissions than the user is supposed to have.

> **Note**
>
> `PassRole` is not an API action in the same way that `RunInstances` or `ListInstanceProfiles` is. Instead, it's a permission that AWS checks whenever a role ARN is passed as a parameter to an API (or the console does this on the user's behalf). It helps an administrator to control which roles can be passed by which users. In this case, it ensures that the user is allowed to attach a specific role to an Amazon EC2 instance.

Example policy that grants a user permission to launch an Amazon EC2 instance with a specific role

The following sample policy allows users to use the Amazon EC2 API to launch an instance with a role. The `Resource` element specifies the Amazon Resource Name (ARN) of a role. By specifying the ARN, the policy grants the user the permission to pass only the `Get-pics` role. If the user tries to specify a different role when launching an instance, the action fails. The user does have permissions to run any instance, regardless of whether they pass a role.

JSON

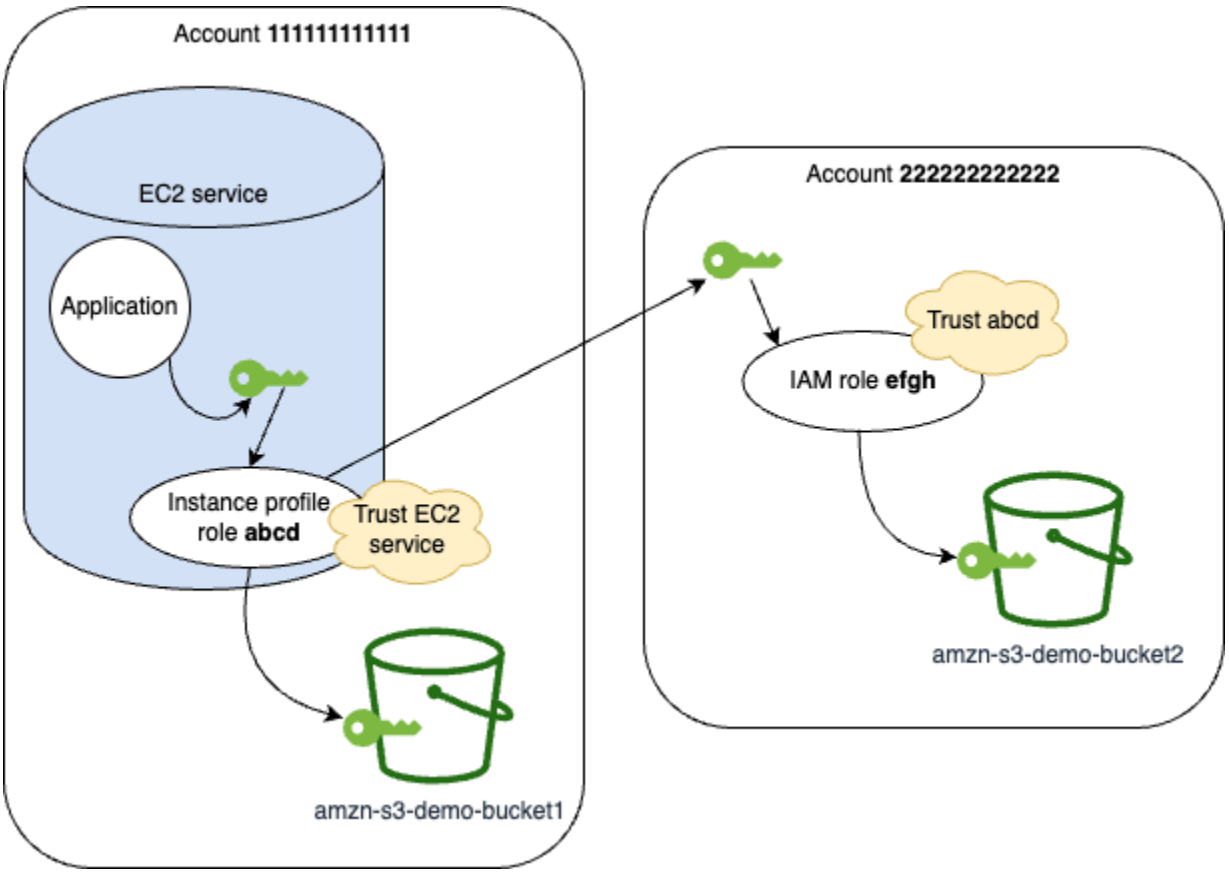
```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/Get-pics"
        }
    ]
}
```

Allowing an instance profile role to switch to a role in another account

You can allow an application running on an Amazon EC2 instance to run commands in another account. To do this, you must allow the Amazon EC2 instance role in the first account to switch to a role in the second account.

Imagine that you are using two AWS accounts and you want to allow an application running on an Amazon EC2 instance to run AWS CLI commands in both accounts. Assume that the Amazon EC2 instance exists in account 111111111111. That instance includes the `abcd` instance profile role that allows the application to perform read-only Amazon S3 tasks on the `amzn-s3-demo-bucket1` bucket within the same 111111111111 account. However, the application must also be allowed to assume the `efgh` cross-account role to access the `amzn-s3-demo-bucket2` Amazon S3 bucket in account 222222222222.



The abcd Amazon EC2 instance profile role must have the following permissions policy to allow the application to access the amzn-s3-demo-bucket1 Amazon S3 bucket:

Account 111111111111 abcd Role Permissions Policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccountLevelS3Actions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
```

```
        },
        {
            "Sid": "AllowListAndReadS3ActionOnMyBucket",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1/*",
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
        },
        {
            "Sid": "AllowIPToAssumeCrossAccountRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::222222222222:role/efgh"
        }
    ]
}
```

The abcd role must trust the Amazon EC2 service to assume the role. To do this, the abcd role must have the following trust policy:

Account 111111111111 abcd Role Trust Policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "abcdTrustPolicy",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"Service": "ec2.amazonaws.com"}
        }
    ]
}
```

Assume that the efgh cross-account role allows read-only Amazon S3 tasks on the amzn-s3-demo-bucket2 bucket within the same 222222222222 account. To do this, the efgh cross-account role must have the following permissions policy:

Account 222222222222 efgh Role Permissions Policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccountLevelS3Actions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "AllowListAndReadS3ActionOnMyBucket",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket2/*",
                "arn:aws:s3:::amzn-s3-demo-bucket2"
            ]
        }
    ]
}
```

The efgh role must trust the abcd instance profile role to assume it. To do this, the efgh role must have the following trust policy:

Account 222222222222 efgh Role Trust Policy

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "efghTrustPolicy",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::111111111111:role/abcd"}
        }
    ]
}
```

How do I get started?

To understand how roles work with Amazon EC2 instances, you need to use the IAM console to create a role, launch an Amazon EC2 instance that uses that role, and then examine the running instance. You can examine the instance metadata to see how the role's temporary credentials are made available to an instance. You can also see how an application that runs on an instance can use the role. Use the following resources to learn more.

-
- SDK walkthroughs. The AWS SDK documentation includes walkthroughs that show an application running on an Amazon EC2 instance that uses temporary credentials for roles to read an Amazon S3 bucket. Each of the following walkthroughs presents similar steps with a different programming language:
  - Configure IAM Roles for Amazon EC2 with the SDK for Java in the *AWS SDK for Java Developer Guide*
  - Launch an Amazon EC2 Instance using the SDK for .NET in the *AWS SDK for .NET Developer Guide*
  - Creating an Amazon EC2 Instance with the SDK for Ruby in the *AWS SDK for Ruby Developer Guide*

Related information

For more information about creating roles or roles for Amazon EC2 instances, see the following information:

- For more information about using IAM roles with Amazon EC2 instances, go to the *Amazon EC2 User Guide*.
- To create a role, see IAM role creation
- For more information about using temporary security credentials, see Temporary security credentials in IAM.
- If you work with the IAM API or CLI, you must create and manage IAM instance profiles. For more information about instance profiles, see Use instance profiles.
- For more information about temporary security credentials for roles in the instance metadata, see Retrieving Security Credentials from Instance Metadata in the *Amazon EC2 User Guide*.

Use instance profiles

Use an instance profile to pass an IAM role to an EC2 instance. For more information, see IAM roles for Amazon EC2 in the *Amazon EC2 User Guide*.

Managing instance profiles (console)

If you use the AWS Management Console to create a role for Amazon EC2, the console automatically creates an instance profile and gives it the same name as the role. When you then use the Amazon EC2 console to launch an instance with an IAM role, you can select a role to associate with the instance. In the console, the list that's displayed is actually a list of instance profile names. The console does not create an instance profile for a role that is not associated with Amazon EC2.

You can use the AWS Management Console to delete IAM roles and instance profiles for Amazon EC2 if the role and the instance profile have the same name. To learn more about deleting instance profiles, see Delete roles or instance profiles.

> **Note**
>
> To update permissions for an instance, replace its instance profile. We do not recommend removing a role from an instance profile, because there is a delay of up to one hour before this change takes effect.

Managing instance profiles (AWS CLI or AWS API)

If you manage your roles from the AWS CLI or the AWS API, you create roles and instance profiles as separate actions. Because roles and instance profiles can have different names, you must know the names of your instance profiles as well as the names of roles they contain. That way you can choose the correct instance profile when you launch an EC2 instance.

You can attach tags to your IAM resources, including instance profiles, to identify, organize, and control access to them. You can tag instance profiles only when you use the AWS CLI or AWS API.

> **Note**
>
> An instance profile can contain only one IAM role, although a role can be included in multiple instance profiles. This limit of one role per instance profile cannot be increased. You can remove the existing role and then add a different role to an instance profile. You must then wait for the change to appear across all of AWS because of eventual consistency. To force the change, you must disassociate the instance profile and then associate the instance profile, or you can stop your instance and then restart it.

Managing instance profiles (AWS CLI)

You can use the following AWS CLI commands to work with instance profiles in an AWS account.

- Create an instance profile: `aws iam create-instance-profile`
- Tag an instance profile: `aws iam tag-instance-profile`
- List tags for an instance profile: `aws iam list-instance-profile-tags`
- Untag an instance profile: `aws iam untag-instance-profile`
- Add a role to an instance profile: `aws iam add-role-to-instance-profile`
- List instance profiles: `aws iam list-instance-profiles`, `aws iam list-instance-profiles-for-role`
- Get information about an instance profile: `aws iam get-instance-profile`
- Remove a role from an instance profile: `aws iam remove-role-from-instance-profile`
- Delete an instance profile: `aws iam delete-instance-profile`

You can also attach a role to an already running EC2 instance by using the following commands. For more information, see IAM Roles for Amazon EC2.

- Attach an instance profile with a role to a stopped or running EC2 instance: `aws ec2 associate-iam-instance-profile`
- Get information about an instance profile attached to an EC2 instance: `aws ec2 describe-iam-instance-profile-associations`
- Detach an instance profile with a role from a stopped or running EC2 instance: `aws ec2 disassociate-iam-instance-profile`

Managing instance profiles (AWS API)

You can call the following AWS API operations to work with instance profiles in an AWS account.

- Create an instance profile: `CreateInstanceProfile`
- Tag an instance profile: `TagInstanceProfile`
- List tags on an instance profile: `ListInstanceProfileTags`
- Untag an instance profile: `UntagInstanceProfile`
- Add a role to an instance profile: `AddRoleToInstanceProfile`
- List instance profiles: `ListInstanceProfiles`, `ListInstanceProfilesForRole`
- Get information about an instance profile: `GetInstanceProfile`
- Remove a role from an instance profile: `RemoveRoleFromInstanceProfile`
- Delete an instance profile: `DeleteInstanceProfile`

You can also attach a role to an already running EC2 instance by calling the following operations. For more information, see IAM Roles for Amazon EC2.

- Attach an instance profile with a role to a stopped or running EC2 instance: `AssociateIamInstanceProfile`
- Get information about an instance profile attached to an EC2 instance: `DescribeIamInstanceProfileAssociations`
- Detach an instance profile with a role from a stopped or running EC2 instance: `DisassociateIamInstanceProfile`

Identity providers and federation into AWS

As a best practice, we recommend that you require human users to use federation with an identity provider to access AWS resources instead of creating individual IAM users in your AWS account. With an identity provider (IdP), you can manage your user identities outside of AWS and give these external user identities permissions to use AWS resources in your account. This is useful if your organization already has its own identity system, such as a corporate user directory. It is also useful if you are creating a mobile app or web application that requires access to AWS resources.

> **Note**
>
> You can also manage human users in IAM Identity Center with an external SAML identity provider instead of using SAML federation in IAM. IAM Identity Center federation with an identity provider provides the capability for you to give people access to multiple AWS accounts in your organization and to multiple AWS applications. For information about specific situations where an IAM user is required, see When to create an IAM user (instead of a role).
>
> You can also use account access manager — an IAM feature that lets you assign existing IAM roles across your organization's accounts to IAM Identity Center users and groups. You can use it alongside permission sets or on its own.

If you prefer to use a single AWS account without enabling IAM Identity Center, you can use IAM with an external IdP that provides identity information to AWS using either OpenID Connect (OIDC) or SAML 2.0 (Security Assertion Markup Language 2.0). OIDC connects applications, like GitHub Actions, that do not run on AWS to AWS resources. Examples of well-known SAML identity providers are Shibboleth and Active Directory Federation Services.

When you use an identity provider, you don't have to create custom sign-in code or manage your own user identities. The IdP provides that for you. Your external users sign in through an IdP, and you can give those external identities permissions to use AWS resources in your account. Identity providers help keep your AWS account secure because you don't have to distribute or embed long-term security credentials, such as access keys, in your application.

Review the following table to help determine which IAM federation type is best for your use case; IAM, IAM Identity Center, or Amazon Cognito. The following summaries and table provide an overview of the methods that your users can employ to gain federated access to AWS resources.

IAM federation type	Account type	Access management of..	Supported identity source
Federation with IAM Identity Center	Multiple accounts managed by AWS Organizations	Your workforce's human users	• SAML 2.0 • Managed Active Directory • Identity Center directory
Federation with IAM	Single, standalone account	• Human users in short-term, small scale deployments • Machine users	• SAML 2.0 • OIDC
Federation with Amazon Cognito identity pools	Any	The users of apps that require IAM authorization to access resources	• SAML 2.0 • OIDC • Select OAuth 2.0 social identity providers

Federation with IAM Identity Center

For centralized access management of human users, we recommend that you use IAM Identity Center to manage access to your accounts and permissions within those accounts. Users in IAM Identity Center are granted short-term credentials to your AWS resources. You can use Active Directory, an external identity provider (IdP), or an IAM Identity Center directory as the identity source for users and groups to assign access to your AWS resources.

IAM Identity Center supports identity federation with SAML (Security Assertion Markup Language) 2.0 to provide federated single sign-on access for users who are authorized to use applications within the AWS access portal. Users can then single sign-on into services that support SAML, including the AWS Management Console and third-party applications, such as Microsoft 365, SAP Concur, and Salesforce.

Federation with IAM

While we strongly recommend managing human users in IAM Identity Center, you can enable federated principal access with IAM for human users in short-term, small scale deployments. IAM allows you to use separate SAML 2.0 and Open ID Connect (OIDC) IdPs and use federated principal attributes for access control. With IAM, you can pass user attributes, such as cost center, title, or locale, from your IdPs to AWS, and implement fine-grained access permissions based on these attributes.

A *workload* is a collection of resources and code that delivers business value, such as an application or backend process. Your workload can require an IAM identity to make requests to AWS services, applications, operational tools, and components. These identities include machines running in your AWS environments, such as Amazon EC2 instances or AWS Lambda functions.

You can also manage machine identities for external parties who need access. To give access to machine identities, you can use IAM roles. IAM roles have specific permissions and provide a way to access AWS by relying on temporary security credentials with a role session. Additionally, you might have machines outside of AWS that need access to your AWS environments. For machines that run outside of AWS you can use IAM Roles Anywhere. For more information about roles, see IAM roles. For details about how to use roles to delegate access across AWS accounts, see IAM tutorial: Delegate access across AWS accounts using IAM roles.

To link an IdP directly to IAM, you create an identity provider entity to establish a trust relationship between your AWS account and the IdP. IAM supports IdPs that are compatible with OpenID Connect (OIDC) or SAML 2.0 (Security Assertion Markup Language 2.0). For more information about using one of these IdPs with AWS, see the following sections:

- OIDC federation
- SAML 2.0 federation

Federation with Amazon Cognito identity pools

Amazon Cognito is designed for developers who want to authenticate and authorize users in their mobile and web apps. Amazon Cognito user pools add sign-in and sign-up features to your app, and identity pools deliver IAM credentials that grant your users access to protected resources that you manage in AWS. Identity pools acquire credentials for temporary sessions through the `AssumeRoleWithWebIdentity` API operation.

Amazon Cognito works with external identity providers that support SAML and OpenID Connect, and with social identity providers like Facebook, Google, and Amazon. Your app can sign in a user with a user pool or an external IdP, then retrieve resources on their behalf with customized temporary sessions in an IAM role.

Additional resources

- For a demonstration on how to create a custom federation proxy that enables single sign-on (SSO) into the AWS Management Console using your organization's authentication system, see Enable custom identity broker access to the AWS console.
- For instructions on enabling your AWS workloads to access external services without storing credentials, see Federating AWS identities to external services.

Common scenarios

> **Note**
>
> We recommend that you require your human users to use temporary credentials when accessing AWS. Have you considered using AWS IAM Identity Center? You can use IAM Identity Center to centrally manage access to multiple AWS accounts and provide users with MFA-protected, single sign-on access to all their assigned accounts from one place. With IAM Identity Center, you can create and manage user identities in IAM Identity Center or easily connect to your existing SAML 2.0 compatible identity provider. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

You can use an external identity provider (IdP) to manage user identities outside of AWS and the external IdP. An external IdP can provide identity information to AWS using either OpenID Connect (OIDC) or Security Assertion Markup Language (SAML). OIDC is commonly used when an application that does not run on AWS needs access to AWS resources.

When you want to configure federation with an external IdP, you create an IAM *identity provider* to inform AWS about the external IdP and its configuration. This establishes trust between your AWS account and the external IdP. The following topics provide common scenarios to use IAM identity providers.

Topics

- Amazon Cognito for mobile apps
- OIDC federation for mobile apps

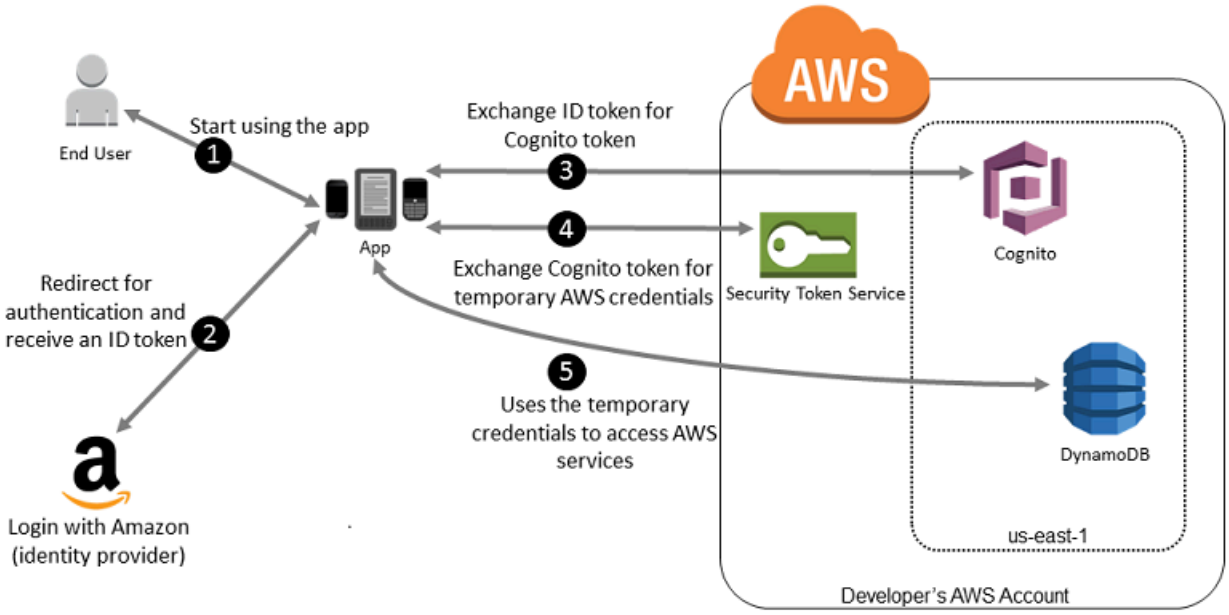
Amazon Cognito for mobile apps

The preferred way to use OIDC federation is to use Amazon Cognito. For example, Adele the developer is building a game for a mobile device where user data such as scores and profiles is stored in Amazon S3 and Amazon DynamoDB. Adele could also store this data locally on the device and use Amazon Cognito to keep it synchronized across devices. She knows that for security and maintenance reasons, long-term AWS security credentials should not be distributed with the game. She also knows that the game might have a large number of users. For all of these reasons, she does not want to create new user identities in IAM for each player. Instead, she builds the game so that users can sign in using an identity that they've already established with a well-known external identity provider (IdP), such as **Login with Amazon**, **Facebook**, **Google**, or any **OpenID Connect** (OIDC)-compatible IdP. Her game can take advantage of the authentication mechanism from one of these providers to validate the user's identity.

To enable the mobile app to access her AWS resources, Adele first registers for a developer ID with her chosen IdPs. She also configures the application with each of these providers. In her AWS account that contains the Amazon S3 bucket and DynamoDB table for the game, Adele uses Amazon Cognito to create IAM roles that precisely define permissions that the game needs. If she is using an OIDC IdP, she also creates an IAM OIDC identity provider entity to establish trust between an Amazon Cognito identity pool in her AWS account and the IdP.

In the app's code, Adele calls the sign-in interface for the IdP that she configured previously. The IdP handles all the details of letting the user sign in, and the app gets an OAuth access token or OIDC ID token from the provider. Adele's app can trade this authentication information for a set of temporary security credentials that consist of an AWS access key ID, a secret access key, and a session token. The app can then use these credentials to access web services offered by AWS. The app is limited to the permissions that are defined in the role that it assumes.

The following figure shows a simplified flow for how this might work, using Login with Amazon as the IdP. For Step 2, the app can also use Facebook, Google, or any OIDC-compatible IdP, but that's not shown here.



1. A customer starts your app on a mobile device. The app asks the user to sign in.
2. The app uses Login with Amazon resources to accept the user's credentials.
3. The app uses the Amazon Cognito API operations `GetId` and `GetCredentialsForIdentity` to exchange the Login with Amazon ID token for an Amazon Cognito token. Amazon Cognito, which has been configured to trust your Login with Amazon project, generates a token that it exchanges for temporary session credentials with AWS STS.
4. The app receives temporary security credentials from Amazon Cognito. Your app can also use the Basic (Classic) workflow in Amazon Cognito to retrieve tokens from AWS STS using `AssumeRoleWithWebIdentity`. For more information, see Identity pools (federated identities) authentication flow in the Amazon Cognito Developer Guide.
5. The temporary security credentials can be used by the app to access any AWS resources required by the app to operate. The role associated with the temporary security credentials and the assigned policies determines what can be accessed.

Use the following process to configure your app to use Amazon Cognito to authenticate users and give your app access to AWS resources. For specific steps to accomplish this scenario, consult the documentation for Amazon Cognito.

1. (Optional) Sign up as a developer with Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC)–compatible IdP and configure one or more apps with the provider. This

step is optional because Amazon Cognito also supports unauthenticated (guest) access for your users.

2. Go to Amazon Cognito in the AWS Management Console. Use the Amazon Cognito wizard to create an identity pool, which is a container that Amazon Cognito uses to keep end user identities organized for your apps. You can share identity pools between apps. When you set up an identity pool, Amazon Cognito creates one or two IAM roles (one for authenticated identities, and one for unauthenticated "guest" identities) that define permissions for Amazon Cognito users.
3. Integrate AWSAmplify with your app, and import the files required to use Amazon Cognito.
4. Create an instance of the Amazon Cognito credentials provider, passing the identity pool ID, your AWS account number, and the Amazon Resource Name (ARN) of the roles that you associated with the identity pool. The Amazon Cognito wizard in the AWS Management Console provides sample code to help you get started.
5. When your app accesses an AWS resource, pass the credentials provider instance to the client object, which passes temporary security credentials to the client. The permissions for the credentials are based on the role or roles that you defined earlier.

For more information, see the following:

- Sign in (Android) in the AWS Amplify Framework Documentation.
- Sign in (iOS) in the AWS Amplify Framework Documentation.

OIDC federation for mobile apps

For best results, use Amazon Cognito as your identity broker for almost all OIDC federation scenarios. Amazon Cognito is easy to use and provides additional capabilities like anonymous (unauthenticated) access, and synchronizing user data across devices and providers. However, if you have already created an app that uses OIDC federation by manually calling the `AssumeRoleWithWebIdentity` API, you can continue to use it and your apps will still work fine.

The process for using OIDC federation ***without*** Amazon Cognito follows this general outline:

1. Sign up as a developer with the external identity provider (IdP) and configure your app with the IdP, who gives you a unique ID for your app. (Different IdPs use different terminology for this process. This outline uses the term *configure* for the process of identifying your app with the IdP.) Each IdP gives you an app ID that's unique to that IdP, so if you configure the same app with

multiple IdPs, your app will have multiple app IDs. You can configure multiple apps with each provider.

The following external links provide information about using some of the commonly used identity providers (IdPs):

- Login with Amazon Developer Center
- Add Facebook Login to Your App or Website on the Facebook developers site.
- Using OAuth 2.0 for Login (OpenID Connect) on the Google developers site.

> **Important**
>
> If you use an OIDC identity provider from Google, Facebook, or Amazon Cognito, do not create a separate IAM identity provider in the AWS Management Console. AWS has these OIDC identity providers built-in and available for your use. Skip the following step and move directly to creating new roles using your identity provider.

2. If you use an IdP other than Google, Facebook or Amazon Cognito compatible with OIDC, then create an IAM identity provider entity for it.
3. In IAM, create one or more roles. For each role, define who can assume the role (the trust policy) and what permissions the app's users have (the permissions policy). Typically, you create one role for each IdP that an app supports. For example, you might create a role assumed by an app if the user signs in through Login with Amazon, a second role for the same app if the user signs in through Facebook, and a third role for the app if the user signs in through Google. For the trust relationship, specify the IdP (like Amazon.com) as the `Principal` (the trusted entity), and include a `Condition` that matches the IdP assigned app ID. Examples of roles for different providers are described in Create a role for a third-party identity provider.
4. In your application, authenticate your users with the IdP. The specifics of how to do this vary both according to which IdP you use (Login with Amazon, Facebook, or Google) and on which platform your app runs. For example, an Android app's method of authentication can differ from that of an iOS app or a JavaScript-based web app.

   Typically, if the user is not already signed in, the IdP takes care of displaying a sign-in page. After the IdP authenticates the user, the IdP returns an authentication token with information about the user to your app. The information included depends on what the IdP exposes and what information the user is willing to share. You can use this information in your app.

5. In your app, make an *unsigned* call to the `AssumeRoleWithWebIdentity` action to request temporary security credentials. In the request, you pass the IdP's authentication token and specify the Amazon Resource Name (ARN) for the IAM role that you created for that IdP. AWS verifies that the token is trusted and valid and if so, returns temporary security credentials to your app that have the permissions for the role that you name in the request. The response also includes metadata about the user from the IdP, such as the unique user ID that the IdP associates with the user.
6. Using the temporary security credentials from the `AssumeRoleWithWebIdentity` response, your app makes signed requests to AWS API operations. The user ID information from the IdP can distinguish users in your app. For example, you can put objects into Amazon S3 folders that include the user ID as prefixes or suffixes. This lets you create access control policies that lock the folder so only the user with that ID can access it. For more information, see AWS STS federated user principals.
7. Your app should cache the temporary security credentials so that you do not have to get new ones each time the app needs to make a request to AWS. By default, the credentials are good for one hour. When the credentials expire (or before then), you make another call to `AssumeRoleWithWebIdentity` to obtain a new set of temporary security credentials. Depending on the IdP and how they manage their tokens, you might have to refresh the IdP's token before you make a new call to `AssumeRoleWithWebIdentity`, since the IdP's tokens also usually expire after a fixed time. If you use the AWS SDK for iOS or the AWS SDK for Android, you can use the AmazonSTSCredentialsProvider action, which manages the IAM temporary credentials, including refreshing them as required.

OIDC federation

Imagine that you are creating an application that accesses AWS resources, such as GitHub Actions that uses workflows to access Amazon S3 and DynamoDB.

When you use these workflows, you make requests to AWS services that must be signed with an AWS access key. However, we **strongly** recommend that you do **not** store AWS credentials long-term in applications outside AWS. Instead, configure your applications to request temporary AWS security credentials dynamically when needed using *OIDC federation*. The supplied temporary credentials map to an AWS role that only has permissions needed to perform the tasks required by the application.

With OIDC federation, you don't need to create custom sign-in code or manage your own user identities. Instead, you can use OIDC in applications, such as GitHub Actions or any other OpenID

Connect (OIDC)-compatible IdP, to authenticate with AWS. They receive an authentication token, known as a JSON Web Token (JWT), and then exchange that token for temporary security credentials in AWS that map to an IAM role with permissions to use specific resources in your AWS account. Using an IdP helps you keep your AWS account secure because you don't have to embed and distribute long-term security credentials with your application.

OIDC federation supports both machine-to-machine authentication (such as CI/CD pipelines, automated scripts, and serverless applications) and human user authentication. For human user authentication scenarios where you need to manage user sign-up, sign-in, and user profiles, consider using Amazon Cognito as an identity broker. For details about using Amazon Cognito with OIDC, see Amazon Cognito for mobile apps.

> **Note**
>
> JSON Web Tokens (JWTs) issued by OpenID Connect (OIDC) identity providers contain an expiration time in the `exp` claim that specifies when the token expires. IAM provides a five-minute window beyond the expiration time specified in the JWT to account for clock skew, as allowed by the OpenID Connect (OIDC) Core 1.0 standard. This means OIDC JWTs received by IAM after the expiration time but within this five-minute window are accepted for further evaluation and processing.

Topics

- Additional resources for OIDC federation
- Create an OpenID Connect (OIDC) identity provider in IAM
- Obtain the thumbprint for an OpenID Connect identity provider
- Identity-provider controls for shared OIDC providers

Additional resources for OIDC federation

The following resources can help you learn more about OIDC federation:

- Use OpenID Connect within your GitHub workflows by Configuring OpenID Connect in Amazon Web Services
- Amazon Cognito Identity in the *Amplify Libraries for Android Guide* and Amazon Cognito Identity in the *Amplify Libraries for Swift Guide*.

- How to use external ID when granting access to your AWS resources on the *AWS Security Blog* provides guidance on securely configuring cross-account access and external identity federation.

Create an OpenID Connect (OIDC) identity provider in IAM

IAM OIDC identity providers are entities in IAM that describe an external identity provider (IdP) service that supports the OpenID Connect (OIDC) standard, such as Google or Salesforce. You use an IAM OIDC identity provider when you want to establish trust between an OIDC-compatible IdP and your AWS account. This is useful when creating a mobile app or web application that requires access to AWS resources, but you don't want to create custom sign-in code or manage your own user identities. For more information about this scenario, see the section called “OIDC federation”.

You can create and manage an IAM OIDC identity provider using the AWS Management Console, the AWS Command Line Interface, the Tools for Windows PowerShell, or the IAM API.

After you create an IAM OIDC identity provider, you must create one or more IAM roles. A role is an identity in AWS that doesn't have its own credentials (as a user does). But in this context, a role is dynamically assigned to an OIDC federated principal that is authenticated by your organization's IdP. The role permits your organization's IdP to request temporary security credentials for access to AWS. The policies assigned to the role determine what users are allowed to do in AWS. To create a role for a third-party identity provider, see Create a role for a third-party identity provider.

> **Important**
>
> When you configure identity-based policies for actions that support `oidc-provider` resources, IAM evaluates the full OIDC identity provider URL, including any specified paths. If your OIDC identity provider URL has a path, you must include that path in the `oidc-provider` ARN as a `Resource` element value. You also have the option to append a forward slash and wildcard (`/*`) to the URL domain or use wildcard characters (`*` and `?`) at any point in the URL path. If the OIDC identity provider URL in the request doesn't match the value set in the policy's `Resource` element, the request fails.

To troubleshoot common issues with IAM OIDC federation, see Resolve errors related to OIDC on AWS re:Post.

Topics

- Prerequisites: Validate configuration of your identity provider

- Creating and managing an OIDC provider (console)
- Creating and managing an IAM OIDC identity provider (AWS CLI)
- Creating and managing an OIDC Identity Provider (AWS API)

Prerequisites: Validate configuration of your identity provider

Before you can create an IAM OIDC identity provider, you must have the following information from your IdP. For more information about obtaining OIDC provider configuration Information, see the documentation for your IdP.

1. Determine your OIDC identity provider's publicly available URL. The URL must begin with https://. Per the OIDC standard, path components are allowed but query parameters are not. Typically, the URL consists of only a hostname, like https://server.example.org or https://example.com. The URL should not contain a port number.
2. Add **/.well-known/openid-configuration** to the end of your OIDC identity provider's URL to see the provider's publicly available configuration document and metadata. You must have a discovery document in JSON format with the provider's configuration document and metadata that can be retrieved from the OpenID Connect provider discovery endpoint URL.
3. Confirm the following values are included in your provider's configuration information. If your openid-configuration is missing any of these fields, you must update your discovery document. This process can vary based on your identity provider, so follow your IdP's documentation to complete this task.

   - issuer: The URL for your domain.
   - jwks_uri: The JSON Web Key Set (JWKS) endpoint where IAM gets your public keys. Your identity provider must include a JSON Web Key Set (JWKS) endpoint in the openid-configuration. This URI defines where to get your public keys that are used to verify the signed tokens from your identity provider.

   > **Note**
   >
   > The JSON Web Key Set (JWKS) must contain at least one key and can have a maximum of 100 RSA keys and 100 EC keys. If your OIDC identity provider's JWKS contains more than 100 RSA keys or 100 EC keys, an `InvalidIdentityToken` exception will be returned when using the AssumeRoleWithWebIdentity API operation with a JWT signed by a key type that exceeds the 100-key limit. For example, if a JWT is signed with the RSA algorithm and there are more than 100

RSA keys in your provider's JWKS, an `InvalidIdentityToken` exception will be returned.

Note

OpenID Connect Discovery on the OpenID Foundation website requires that the JSON Web Key Set (JWKS) at `jwks_uri` must not contain private or symmetric key values. Your JWKS must not include the RSA d, p, q, dp, dq, `qi`, or `oth` parameters, the elliptic curve d parameter, or the k parameter of a symmetric (`oct`) key.
If your JWKS contains any of these values, AWS STS returns an `InvalidIdentityToken` exception when you use the AssumeRoleWithWebIdentity API operation. The exception message is "Couldn't retrieve verification key from your identity provider, please reference AssumeRoleWithWebIdentity documentation for requirements". AWS STS also returns this message for other key retrieval failures.
You must treat any private key that you published as compromised. Republishing a clean JWKS alone is not sufficient. To remediate this exposure, complete the following steps:

- Generate a new key pair and configure your identity provider to sign tokens with the new key ID (`kid`).
- Remove the exposed key from your JWKS.
- Review your AWS CloudTrail `AssumeRoleWithWebIdentity` events for the period when your JWKS exposed the private key.
- Revoke active sessions for roles that trust the provider. For more information, see Revoke IAM role temporary security credentials.

- claims_supported: Information about the user that helps you ensure OIDC authentication responses from your IdP contain the required attributes AWS uses in IAM policies to check permissions for OIDC federated principals. For a list of IAM condition keys that can be used for claims, see Available keys for AWS OIDC federation.
  - aud: You must determine the audience claim value your IdP issues in JSON Web Tokens (JWTs). The audience (aud) claim is application specific and identifies the intended recipients of the token. When you register a mobile or web app with an OpenID Connect provider, they establish a client ID that identifies the application. The client ID is a unique

identifier for your app that is passed in the aud claim for authentication. The aud claim must match the Audience value when creating your IAM OIDC identity provider.

- iat: Claims must include a value for `iat` that represents the time that the ID token is issued.
- iss: The URL of the identity provider. The URL must begin with https:// and should correspond to the Provider URL provided to IAM. Per the OIDC standard, path components are allowed but query parameters are not. Typically, the URL consists of only a hostname, like https://server.example.org or https://example.com. The URL should not contain a port number.

- response_types_supported: id_token
- subject_types_supported: public
- id_token_signing_alg_values_supported: RS256, RS384, RS512, ES256, ES384, ES512

Note

You can include additional claims like `my_custom_claim` in the example below; however, AWS STS will ignore the claim.

```
{
  "issuer": "https://example-domain.com",
  "jwks_uri": "https://example-domain.com/jwks/keys",
  "claims_supported": [
    "aud",
    "iat",
    "iss",
    "name",
    "sub",
    "my_custom_claim"
  ],
  "response_types_supported": [
    "id_token"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256",
    "RS384",
    "RS512",
    "ES256",
    "ES384",
    "ES512"
```

```
  ],
  "subject_types_supported": [
    "public"
  ]
}
```

Creating and managing an OIDC provider (console)

Follow these instructions to create and manage an IAM OIDC identity provider in the AWS Management Console.

> **Important**
>
> If you are using an OIDC identity provider from either Google, Facebook, or Amazon Cognito, do not create a separate IAM identity provider using this procedure. These OIDC identity providers are already built-in to AWS and are available for your use. Instead, follow the steps to create new roles for your identity provider, see Create a role for OpenID Connect federation (console).

To create an IAM OIDC identity provider (console)

1. Before you create an IAM OIDC identity provider, you must register your application with the IdP to receive a *client ID*. The client ID (also known as *audience*) is a unique identifier for your app that is issued to you when you register your app with the IdP. For more information about obtaining a client ID, see the documentation for your IdP.

   > **Note**
   >
   > AWS secures communication with OIDC identity providers (IdPs) using our library of trusted root certificate authorities (CAs) to verify the JSON Web Key Set (JWKS) endpoint's TLS certificate. If your OIDC IdP relies on a certificate that is not signed by one of these trusted CAs, only then we secure communication using the thumbprints set in the IdP's configuration. AWS will fall back to thumbprint verification if we are unable to retrieve the TLS certificate or if TLS v1.3 is required.

2. Open the IAM console at https://console.aws.amazon.com/iam/.

3. In the navigation pane, choose **Identity providers**, and then choose **Add provider**.
4. For **Configure provider**, choose **OpenID Connect**.
5. For **Provider URL**, type the URL of the IdP. The URL must comply with these restrictions:
   - The URL is case-sensitive.
   - The URL must begin with **`https://`**.
   - The URL should not contain a port number.
   - Within your AWS account, each IAM OIDC identity provider must use a unique URL. If you try to submit a URL that has already been used for an OpenID Connect provider in the AWS account, you will get an error.
6. For **Audience**, type the client ID of the application that you registered with the IdP and received in Step 1, and that make requests to AWS. If you have additional client IDs (also known as *audiences*) for this IdP, you can add them later on the provider detail page.

   > **Note**
   >
   > If your IdP JWT token includes the `azp` claim, enter this value as the Audience value. If your OIDC identity provider is setting both `aud` and `azp` claims in the token, AWS STS will use the value in the `azp` claim as the `aud` claim.

7. (Optional) For **Add tags**, you can add key–value pairs to help you identify and organize your IdPs. You can also use tags to control access to AWS resources. To learn more about tagging IAM OIDC identity providers, see Tag OpenID Connect (OIDC) identity providers. Choose **Add tag**. Enter values for each tag key-value pair.
8. Verify the information that you have provided. When you are done choose **Add provider**. IAM will attempt to retrieve and use the top intermediate CA thumbprint of the OIDC IdP server certificate to create the IAM OIDC identity provider.

   > **Note**
   >
   > The OIDC identity provider's certificate chain must start with the domain or issuer URL, then the intermediate certificate, and end with the root certificate. If the certificate chain order is different or includes duplicate or additional certificates, then you receive a signature mismatch error and STS fails to validate the JSON Web Token (JWT). Correct the order of the certificates in the chain returned from the server to resolve the

error. For more information about certificate chain standards, see certificate_list in RFC 5246 on the RFC Series website.

9. Assign an IAM role to your identity provider to give external user identities managed by your identity provider permissions to access AWS resources in your account. To learn more about creating roles for identity federation, see Create a role for a third-party identity provider.

> **Note**
>
> OIDC IdPs used in a role trust policy must be in the same account as the role that trusts it.

To add or remove a thumbprint for an IAM OIDC identity provider (console)

> **Note**
>
> AWS secures communication with OIDC identity providers (IdPs) using our library of trusted root certificate authorities (CAs) to verify the JSON Web Key Set (JWKS) endpoint's TLS certificate. If your OIDC IdP relies on a certificate that is not signed by one of these trusted CAs, only then we secure communication using the thumbprints set in the IdP's configuration. AWS will fall back to thumbprint verification if we are unable to retrieve the TLS certificate or if TLS v1.3 is required.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers**. Then choose the name of the IAM identity provider that you want to update.
3. Choose the **Endpoint verification** tab, then in the **Thumbprints** section, choose **Manage**. To enter a new thumbprint value, choose **Add thumbprint**. To remove a thumbprint, choose **Remove** next to the thumbprint that you want to remove.

> **Note**
>
> An IAM OIDC identity provider must have at least one and can have a maximum of five thumbprints.

When you are done, choose **Save changes**.

To add an audience for an IAM OIDC identity provider (console)

1. In the navigation pane, choose **Identity providers**, then choose the name of the IAM identity provider that you want to update.
2. In the **Audiences** section, choose **Actions** and select **Add audience**.
3. Type the client ID of the application that you registered with the IdP and received in Step 1, and that will make requests to AWS. Then choose **Add audiences**.

> **Note**
>
> An IAM OIDC identity provider must have at least one and can have a maximum of 100 audiences.

To remove an audience for an IAM OIDC identity provider (console)

1. In the navigation pane, choose **Identity providers**, then choose the name of the IAM identity provider that you want to update.
2. In the **Audiences** section, select the radio button next to the audience that you want to remove, then select **Actions**.
3. Choose **Remove audience**. A new window opens.
4. If you remove an audience, identities federating with the audience cannot assume roles associated with the audience. In the window, read the warning and confirm that you want to remove the audience by typing the word `remove` in the field.
5. Choose **Remove** to remove the audience.

To delete an IAM OIDC identity provider (console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers**.
3. Select the checkbox next to the IAM identity provider that you want to delete. A new window opens.

4. Confirm that you want to delete the provider by typing the word `delete` in the field. Then, choose **Delete**.

Creating and managing an IAM OIDC identity provider (AWS CLI)

You can use the following AWS CLI commands to create and manage IAM OIDC identity providers.

To create an IAM OIDC identity provider (AWS CLI)

1. (Optional) To get a list of all the IAM OIDC identity providers in your AWS account, run the following command:
   - `aws iam list-open-id-connect-providers`
2. To create a new IAM OIDC identity provider, run the following command:
   - `aws iam create-open-id-connect-provider`

To update the list of server certificate thumbprints for an existing IAM OIDC identity provider (AWS CLI)

- To update the list of server certificate thumbprints for an IAM OIDC identity provider, run the following command:
  - `aws iam update-open-id-connect-provider-thumbprint`

To tag an existing IAM OIDC identity provider (AWS CLI)

- To tag an existing IAM OIDC identity provider, run the following command:
  - `aws iam tag-open-id-connect-provider`

To list tags for an existing IAM OIDC identity provider (AWS CLI)

- To list tags for an existing IAM OIDC identity provider, run the following command:
  - `aws iam list-open-id-connect-provider-tags`

To remove tags on an IAM OIDC identity provider (AWS CLI)

- To remove tags on an existing IAM OIDC identity provider, run the following command:
  - `aws iam untag-open-id-connect-provider`

To add or remove a client ID from an existing IAM OIDC identity provider (AWS CLI)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, run the following command:
   - `aws iam list-open-id-connect-providers`
2. (Optional) To get detailed information about an IAM OIDC identity provider, run the following command:
   - `aws iam get-open-id-connect-provider`
3. To add a new client ID to an existing IAM OIDC identity provider, run the following command:
   - `aws iam add-client-id-to-open-id-connect-provider`
4. To remove a client from an existing IAM OIDC identity provider, run the following command:
   - `aws iam remove-client-id-from-open-id-connect-provider`

To delete an IAM OIDC identity provider (AWS CLI)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, run the following command:
   - `aws iam list-open-id-connect-providers`
2. (Optional) To get detailed information about an IAM OIDC identity provider, run the following command:
   - `aws iam get-open-id-connect-provider`
3. To delete an IAM OIDC identity provider, run the following command:
   - `aws iam delete-open-id-connect-provider`

Creating and managing an OIDC Identity Provider (AWS API)

You can use the following IAM API commands to create and manage OIDC providers.

To create an IAM OIDC identity provider (AWS API)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, call the following operation:
   - `ListOpenIDConnectProviders`
2. To create a new IAM OIDC identity provider, call the following operation:
   - `CreateOpenIDConnectProvider`

To update the list of server certificate thumbprints for an existing IAM OIDC identity provider (AWS API)

- To update the list of server certificate thumbprints for an IAM OIDC identity provider, call the following operation:
  - `UpdateOpenIDConnectProviderThumbprint`

To tag an existing IAM OIDC identity provider (AWS API)

- To tag an existing IAM OIDC identity provider, call the following operation:
  - `TagOpenIDConnectProvider`

To list tags for an existing IAM OIDC identity provider (AWS API)

- To list tags for an existing IAM OIDC identity provider, call the following operation:
  - `ListOpenIDConnectProviderTags`

To remove tags on an existing IAM OIDC identity provider (AWS API)

- To remove tags on an existing IAM OIDC identity provider, call the following operation:
  - `UntagOpenIDConnectProvider`

To add or remove a client ID from an existing IAM OIDC identity provider (AWS API)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, call the following operation:
   - `ListOpenIDConnectProviders`
2. (Optional) To get detailed information about an IAM OIDC identity provider, call the following operation:
   - `GetOpenIDConnectProvider`
3. To add a new client ID to an existing IAM OIDC identity provider, call the following operation:
   - `AddClientIDToOpenIDConnectProvider`
4. To remove a client ID from an existing IAM OIDC identity provider, call the following operation:
   - `RemoveClientIDFromOpenIDConnectProvider`

To delete an IAM OIDC identity provider (AWS API)

1. (Optional) To get a list of all the IAM OIDC identity provider in your AWS account, call the following operation:
   - `ListOpenIDConnectProviders`
2. (Optional) To get detailed information about an IAM OIDC identity provider, call the following operation:
   - `GetOpenIDConnectProvider`
3. To delete an IAM OIDC identity provider, call the following operation:
   - `DeleteOpenIDConnectProvider`

Obtain the thumbprint for an OpenID Connect identity provider

When you create an OpenID Connect (OIDC) identity provider in IAM, IAM requires the thumbprint for the top intermediate certificate authority (CA) that signed the certificate used by the external identity provider (IdP). The thumbprint is a signature for the CA's certificate that was used to issue the certificate for the OIDC-compatible IdP. When you create an IAM OIDC identity provider, you are trusting identities authenticated by that IdP to have access to your AWS account. By using the

CA's certificate thumbprint, you trust any certificate issued by that CA with the same DNS name as the one registered. This eliminates the need to update trusts in each account when you renew the IdP's signing certificate.

Important

In most cases, the federation server uses two different certificates:

- The first establishes an HTTPS connection between AWS and your IdP. This should be issued by a well-known public root CA, such as AWS Certificate Manager. This enables the client to check the reliability and status of the certificate.
- The second is used to encrypt tokens, and should be signed by a private or public *root* CA.

You can create an IAM OIDC identity provider with the AWS Command Line Interface, the Tools for Windows PowerShell, or the IAM API. When you use these methods, you have the option to manually provide a thumbprint. If you choose not to include a thumbprint, IAM will retrieve the top intermediate CA thumbprint of the OIDC IdP server certificate. If you choose to include a thumbprint, you must obtain the thumbprint manually and supply it to AWS.

When you create an OIDC identity provider with the IAM console, IAM attempts to retrieve the top intermediate CA thumbprint of the OIDC IdP server certificate for you.

We recommend that you also obtain the thumbprint for your OIDC IdP manually and verify that IAM retrieved the correct thumbprint. For more information about obtaining certificate thumbprints, see the following sections.

Note

AWS secures communication with OIDC identity providers (IdPs) using our library of trusted root certificate authorities (CAs) to verify the JSON Web Key Set (JWKS) endpoint's TLS certificate. If your OIDC IdP relies on a certificate that is not signed by one of these trusted CAs, only then we secure communication using the thumbprints set in the IdP's configuration. AWS will fall back to thumbprint verification if we are unable to retrieve the TLS certificate or if TLS v1.3 is required.

Important

If your OIDC identity provider uses different hosts for the discovery endpoint (the issuer URL) and the JWKS endpoint (`jwks_uri`), you must include the thumbprint for both endpoints in your OIDC provider's thumbprint list. AWS validates the thumbprint of both endpoints when falling back to thumbprint verification. If either thumbprint is missing from the configured list, authentication fails.
Most OIDC providers host both endpoints on the same domain, so a single thumbprint covers both. You only need to add a separate thumbprint if your provider uses different hosts or certificates for the two endpoints.

Obtain certificate thumbprint

You use a web browser and the OpenSSL command line tool to obtain the certificate thumbprint for an OIDC provider. However, you do not need to manually obtain the certificate thumbprint to create an IAM OIDC identity provider. You can use the following procedure to obtain the certificate thumbprint of your OIDC provider.

To obtain the thumbprint for an OIDC IdP

1. Before you can obtain the thumbprint for an OIDC IdP, you need to obtain the OpenSSL command line tool. You use this tool to download the OIDC IdP certificate chain and produce a thumbprint of the final certificate in the certificate chain. If you need to install and configure OpenSSL, follow the instructions at Install OpenSSL and Configure OpenSSL.
2. Start with the OIDC IdP URL (for example, `https://server.example.com`), and then add `/.well-known/openid-configuration` to form the URL for the IdP's configuration document, such as the following:

   `https://server.example.com/.well-known/openid-configuration`

   Open this URL in a web browser, replacing *`server.example.com`* with your IdP server name.
3. In the displayed document, use your web browser **Find** feature to locate the text `"jwks_uri"`. Immediately following the text `"jwks_uri"`, there is a colon (:) followed by a URL. Copy the fully qualified domain name of the URL. Do not include `https://` or any path that comes after the top-level domain.

   ```
   {
   ```

```
 "issuer": "https://accounts.example.com",
 "authorization_endpoint": "https://accounts.example.com/o/oauth2/v2/auth",
 "device_authorization_endpoint": "https://oauth2.exampleapis.com/device/code",
 "token_endpoint": "https://oauth2.exampleapis.com/token",
 "userinfo_endpoint": "https://openidconnect.exampleapis.com/v1/userinfo",
 "revocation_endpoint": "https://oauth2.exampleapis.com/revoke",
 "jwks_uri": "https://www.exampleapis.com/oauth2/v3/certs",
...
```

Note

If your discovery and JWKS endpoints use different hosts, note the fully qualified domain name from your issuer URL (the URL you used in Step 2, without the `/.well-known/openid-configuration` path). If this domain differs from the `jwks_uri` domain you copied in this step, perform Step 4 through Step 6 for both domains and include both thumbprints when creating your OIDC provider.

4. Use the OpenSSL command line tool to run the following command. Replace *keys.example.com* with the domain name you obtained in Step 3.

```
openssl s_client -servername keys.example.com -showcerts -
connect keys.example.com:443
```

5. In your command window, scroll up until you see a certificate similar to the following example. If you see more than one certificate, find the last certificate displayed (at the end of the command output). This contains the certificate of the top intermediate CA in the certificate authority chain.

```
-----BEGIN CERTIFICATE-----
 MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
 VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
 b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
 BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN
 MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD
 VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z
 b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt
 YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ
 21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
 rDHudUZg3qX4waLG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
 Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
```

```
nUhVVxYUntneD9+h8Mg9q6q+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----
```

Copy the certificate (including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` lines) and paste it into a text file. Then save the file with the file name **`certificate.crt`**.

> **Note**
>
> The OIDC identity provider's certificate chain must start with the domain or issuer URL, include any intermediate certificates (if present), and end with the root certificate. If the certificate chain order is different or includes duplicate or additional certificates, you will receive a signature mismatch error and STS fails to validate the JSON Web Token (JWT). Correct the order of the certificates in the chain returned from the server to resolve the error. For more information about certificate chain standards, see certificate_list in RFC 5246 on the RFC Series website.

6. Use the OpenSSL command line tool to run the following command.

```
openssl x509 -in certificate.crt -fingerprint -sha1 -noout
```

Your command window displays the certificate thumbprint, which looks similar to the following example:

```
SHA1 Fingerprint=99:0F:41:93:97:2F:2B:EC:F1:2D:DE:DA:52:37:F9:C9:52:F2:0D:9E
```

Remove the colon characters (:) from this string to produce the final thumbprint, like this:

```
990F4193972F2BECF12DDEDA5237F9C952F20D9E
```

> **Note**
>
> If your discovery endpoint uses a different host than your JWKS endpoint, repeat Step 4 through Step 6 using the discovery endpoint domain (your issuer URL domain).

Include both thumbprints when creating or updating your OIDC identity provider using CreateOpenIDConnectProvider or UpdateOpenIDConnectProviderThumbprint.

7. If you are creating the IAM OIDC identity provider with the AWS CLI, Tools for Windows PowerShell, or the IAM API, providing a thumbprint is optional. If you choose not to include a thumbprint during creation, IAM will retrieve the top intermediate CA thumbprint of the OIDC IdP server certificate. After the IAM OIDC identity provider is created, you can compare this thumbprint to the thumbprint retrieved by IAM.

   If you are creating the IAM OIDC identity provider in the IAM console, the console attempts to retrieve the top intermediate CA thumbprint of the OIDC IdP server certificate for you. You can compare this thumbprint to the thumbprint retrieved by IAM. After the IAM OIDC identity provider is created, you can view the thumbprint for the IAM OIDC identity provider in the **Endpoint verification** tab on the OIDC provider **Summary** console page.

   Important

   If the thumbprint you obtained does not match the one you see in the IAM OIDC identity provider thumbprint details, you should not use the OIDC provider. Instead, you should delete the created OIDC provider and then try again to create the OIDC provider after some time has passed. Verify that the thumbprints match before you use the provider. If the thumbprints still do not match after a second attempt, use the IAM Forum to contact AWS.

Install OpenSSL

If you don't already have OpenSSL installed, follow the instructions in this section.

To install OpenSSL on Linux or Unix

1. Go to OpenSSL: Source, Tarballs (https://openssl.org/source/).
2. Download the latest source and build the package.

To install OpenSSL on Windows

1. Go to OpenSSL: Binary Distributions (https://wiki.openssl.org/index.php/Binaries) for a list of sites from which you can install the Windows version.

2. Follow the instructions on your selected site to start the installation.
3. If you are asked to install the **Microsoft Visual C++ 2008 Redistributables** and it is not already installed on your system, choose the download link appropriate for your environment. Follow the instructions provided by the **Microsoft Visual C++ 2008 Redistributable Setup Wizard**.

   > **Note**
   >
   > If you are not sure whether the Microsoft Visual C++ 2008 Redistributables is already installed on your system, you can try installing OpenSSL first. The OpenSSL installer displays an alert if the Microsoft Visual C++ 2008 Redistributables is not yet installed. Make sure that you install the architecture (32-bit or 64-bit) that matches the version of OpenSSL that you install.

4. After you have installed the Microsoft Visual C++ 2008 Redistributables, select the appropriate version of the OpenSSL binaries for your environment and save the file locally. Start the **OpenSSL Setup Wizard**.
5. Follow the instructions described in the **OpenSSL Setup Wizard**.

Configure OpenSSL

Before you use OpenSSL commands, you must configure the operating system so that it has information about the location where OpenSSL is installed.

To configure OpenSSL on Linux or Unix

1. At the command line, set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

   ```
   $ export OpenSSL_HOME=path_to_your_OpenSSL_installation
   ```

2. Set the path to include the OpenSSL installation:

   ```
   $ export PATH=$PATH:$OpenSSL_HOME/bin
   ```

> **Note**
>
> Any changes you make to environment variables with the `export` command are valid only for the current session. You can make persistent changes to the environment variables by setting them in your shell configuration file. For more information, see the documentation for your operating system.

To configure OpenSSL on Windows

1. Open a **Command Prompt** window.
2. Set the `OpenSSL_HOME` variable to the location of the OpenSSL installation:

```
C:\> set OpenSSL_HOME=path_to_your_OpenSSL_installation
```

3. Set the `OpenSSL_CONF` variable to the location of the configuration file in your OpenSSL installation:

```
C:\> set OpenSSL_CONF=path_to_your_OpenSSL_installation\bin\openssl.cfg
```

4. Set the path to include the OpenSSL installation:

```
C:\> set Path=%Path%;%OpenSSL_HOME%\bin
```

> **Note**
>
> Any changes you make to Windows environment variables in a **Command Prompt** window are valid only for the current command line session. You can make persistent changes to the environment variables by setting them as system properties. The exact procedures depend on what version of Windows you're using. (For example, in Windows 7, open **Control Panel**, **System and Security**, **System**. Then choose **Advanced system settings**, **Advanced** tab, **Environment Variables**.) For more information, see the Windows documentation.

Identity-provider controls for shared OIDC providers

For recognized shared OpenID Connect (OIDC) identity providers (IdPs), IAM requires explicit evaluation of specific claims in role trust policies. These required claims, called *identity-provider controls*, are evaluated by IAM during role creation and trust policy updates. If the role trust policy does not evaluate the controls required by the shared OIDC IdP, the role creation or update would fail. This ensures that only authorized identities from the intended organization can assume roles and access AWS resources. This security control is crucial when OIDC providers are shared across multiple AWS customers.

Identity-provider controls will not be evaluated by IAM for existing OIDC role trust policies. For any modifications to the role trust policy for existing OIDC roles, IAM will require that identity-provider controls be included in the role trust policy.

OIDC provider types

IAM categorizes OIDC identity providers into two distinct types: **private** and **shared**. A private OIDC IdP can be owned and managed by a single organization or can be a tenant of a SaaS provider, with its OIDC Issuer URL serving as a unique identifier specific to that organization. In contrast, a shared OIDC IdP is utilized across multiple organizations, where the OIDC Issuer URL might be identical for all organizations using that shared identity provider.

The table below outlines the key differences between private and shared OIDC providers:

Characteristic	Private OIDC Provider	Shared OIDC Provider
Issuer	Unique to the organization	Shared across multiple organizations
Tenancy Information	Communicated through unique Issuer	Communicated through claims in JWT
Trust Policy Requirements	No specific claim evaluation required	Evaluation of specific claims required

Shared OIDC identity providers with identity-provider controls

When you create or modify an OIDC provider in IAM, the system automatically identifies and evaluates required claims for recognized shared OIDC providers. If identity-provider

controls are not configured in the role trust policy, the role creation or update will fail with a MalformedPolicyDocument error.

The following table lists the shared OIDC providers that require identity-provider controls in role trust policies and additional information to help you configure identity-provider controls.

OIDC IdP	OIDC URL	Tenancy Claim	Required Claims
Amazon Cognito	`cognito-identity.amazonaws.com`	aud	`cognito-identity.amazonaws.com:aud`
Azure Sentinel	`https://sts.windows.net/33e01921-4d64-4f8c-a055-5bdaffd5e33d`	`sts:RoleSessionName`	`sts:RoleSessionName`
Buildkite	`https://agent.buildkite.com`	sub	`agent.buildkite.com:sub`
Codefresh SaaS	`https://oidc.codefresh.io`	sub	`oidc.codefresh.io:sub`
DVC Studio	`https://studio.datachain.ai/api`	sub	`studio.datachain.ai/api:sub`
GitHub actions	`https://token.actions.githubusercontent.com`	sub or job_workflow_ref	`token.actions.githubusercontent.com:sub` or

OIDC IdP	OIDC URL	Tenancy Claim	Required Claims
			token.actions.githubusercontent.com:job_workflow_ref
GitHub audit log streaming	https://oidc-configuration.audit-log.githubusercontent.com	sub	oidc-configuration.audit-log.githubusercontent.com:sub
GitHub vstoken	https://vstoken.actions.githubusercontent.com	sub	vstoken.actions.githubusercontent.com:sub
GitLab	https://gitlab.com	sub	gitlab.com:sub

OIDC IdP	OIDC URL	Tenancy Claim	Required Claims
IBM Turbonomic SaaS*	• https://rh-oidc.s3.us-east-1.amazonaws.com/22ejnvnnturfmt6km08iidd0nt4hekbn7 • https://rh-oidc.s3.us-east-1.amazonaws.com/23e3sd27sju1hoou6ohfs68vbno607tr • https://rh-oidc.s3.us-east-1.amazonaws.com/23ne21h005qjl3n33d8dui5dlrmv2tmg • https://rh-oidc.s3.us-east-1.amazonaws.com/24jrf12m5dj7ljlfb4ta2frhrcoadm26	sub	• rh-oidc.s3.us-east-1.amazonaws.com/22ejnvnnturfmt6km08iidd0nt4hekbn7:sub • rh-oidc.s3.us-east-1.amazonaws.com/23e3sd27sju1hoou6ohfs68vbno607tr:sub • rh-oidc.s3.us-east-1.amazonaws.com/23ne21h005qjl3n33d8dui5dlrmv2tmg:sub • rh-oidc.s3.us-east-1.amazonaws.com/24jrf12m5dj7ljlfb4ta2frhrcoadm26:sub

OIDC IdP	OIDC URL	Tenancy Claim	Required Claims
	• https://oidc.op1.openshiftapps.com/2f785sojlpb85i7402pk3qogugim5nfb • https://oidc.op1.openshiftapps.com/2c51blsaqa9gkjt0o9rt11mle8mmropu		• oidc.op1.openshiftapps.com/2f785sojlpb85i7402pk3qogugim5nfb:sub • oidc.op1.openshiftapps.com/2c51blsaqa9gkjt0o9rt11mle8mmropu:sub
Pulumi Cloud	https://api.pulumi.com/oidc	aud	api.pulumi.com/oidc:aud
sandboxes.cloud	https://sandboxes.cloud	aud	sandboxes.cloud:aud
Scalr	https://scalr.io	sub	scalr.io:sub
Shisho Cloud	https://tokens.cloud.shisho.dev	sub	tokens.cloud.shisho.dev:sub
Terraform Cloud	https://app.terraform.io	sub	app.terraform.io:sub

OIDC IdP	OIDC URL	Tenancy Claim	Required Claims
Upbound	`https://proidc.upbound.io`	sub	`proidc.upbound.io:sub`
Vercel global endpoint	`https://oidc.vercel.com`	aud	`oidc.vercel.com:aud`

* IBM Turbonomic periodically updates their OIDC Issuer URL with new versions of the platform. We will add additional Turbonomic OIDC issuers in scope as a shared provider as needed.

For any new OIDC IdPs that IAM identifies as shared, the required identity-provider controls for role trust policies will be documented and enforced in a similar manner.

Additional resources

Additional resources:

- For more information about creating an IAM role for OIDC federation, see Create a role for OpenID Connect federation (console).
- For a list of IAM condition keys that can be used for claims, see Available keys for AWS OIDC federation.

SAML 2.0 federation

AWS supports identity federation with SAML 2.0 (Security Assertion Markup Language 2.0), an open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call AWS API operations without you having to create an IAM user for everyone in your organization. By using SAML, you can simplify the process of configuring federation with AWS, because you can use the IdP's service instead of writing custom identity proxy code.

> **Note**
>
> IAM SAML identity federation supports encrypted SAML responses from SAML-based federated identity providers (IdPs). IAM Identity Center and Amazon Cognito do not support encrypted SAML assertions from IAM SAML identity providers.

You can indirectly add support for encrypted SAML assertions to Amazon Cognito identity pool federation with Amazon Cognito user pools. User pools have SAML federation that's independent of IAM SAML federation and supports SAML signing and encryption. Although this feature doesn't extend directly to identity pools, user pools can be IdPs to identity pools. To use SAML encryption with identity pools, add a SAML provider with encryption to a user pool that is an IdP to an identity pool.
Your SAML provider must be able to encrypt SAML assertions with a key that your user pool provides. User pools won't accept assertions encrypted with a certificate that IAM has provided.

IAM federation supports these use cases:

- **Federated access to allow a user or application in your organization to call AWS API operations**. This use case is discussed in the following section. You use a SAML assertion (as part of the authentication response) that is generated in your organization to get temporary security credentials. This scenario is similar to other federation scenarios that IAM supports, like those described in Request temporary security credentials and OIDC federation. However, SAML 2.0–based IdPs in your organization handle many of the details at run time for performing authentication and authorization checking.
- **Web-based single sign-on (SSO) to the AWS Management Console from your organization**. Users can sign in to a portal in your organization hosted by a SAML 2.0–compatible IdP, select an option to go to AWS, and be redirected to the console without having to provide additional sign-in information. You can use a third-party SAML IdP to establish SSO access to the console or you can create a custom IdP to enable console access for your external users. For more information about building a custom IdP, see Enable custom identity broker access to the AWS console.

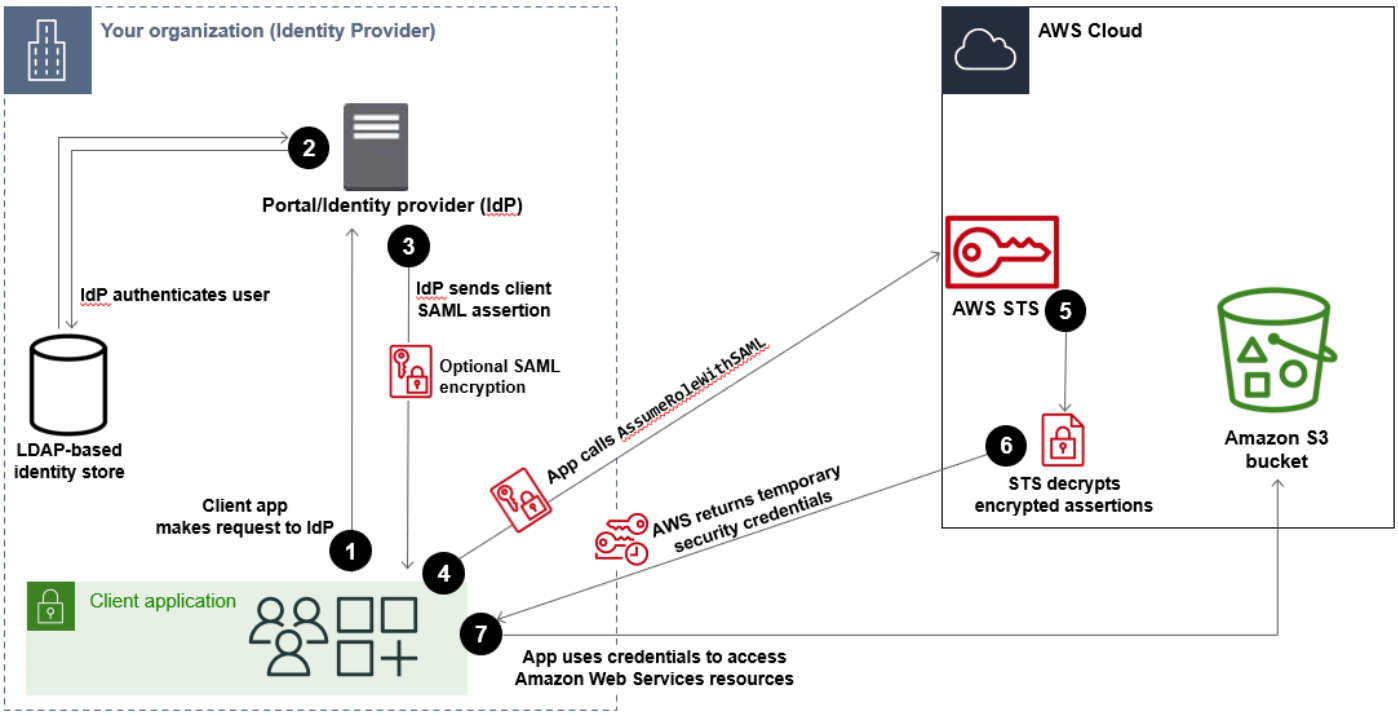
Topics

- Using SAML-based federation for API access to AWS
- Overview of configuring SAML 2.0-based federation
- Overview of the role to allow SAML-federated access to your AWS resources
- Uniquely identifying users in SAML-based federation
- Create a SAML identity provider in IAM
- Configure your SAML 2.0 IdP with relying party trust and adding claims
- Integrate third-party SAML solution providers with AWS

- Configure SAML assertions for the authentication response
- Enabling SAML 2.0 federated principals to access the AWS Management Console
- View a SAML response in your browser

Using SAML-based federation for API access to AWS

Assume that you want to provide a way for employees to copy data from their computers to a backup folder. You build an application that users can run on their computers. On the back end, the application reads and writes objects in an Amazon S3 bucket. Users don't have direct access to AWS. Instead, the following process is used:



1. A user in your organization uses a client app to request authentication from your organization's IdP.
2. The IdP authenticates the user against your organization's identity store.
3. The IdP constructs a SAML assertion with information about the user and sends the assertion to the client app. When you enable SAML encryption for your IAM SAML IdP, this assertion is encrypted by your external IdP.
4. The client app calls the AWS STS `AssumeRoleWithSAML` API, passing the ARN of the SAML provider, the ARN of the role to assume, and the SAML assertion from IdP. If encryption is enabled, the assertion passed through the client app remains encrypted in transit.

5. (Optional) AWS STS uses the private key you uploaded from your external IdP to decrypt the encrypted SAML assertion.
6. The API response to the client app includes temporary security credentials.
7. The client app uses the temporary security credentials to call Amazon S3 API operations.

Overview of configuring SAML 2.0-based federation

Before you can use SAML 2.0-based federation as described in the preceding scenario and diagram, you must configure your organization's IdP and your AWS account to trust each other. The general process for configuring this trust is described in the following steps. Inside your organization, you must have an IdP that supports SAML 2.0, like Microsoft Active Directory Federation Service (AD FS, part of Windows Server), Shibboleth, or another compatible SAML 2.0 provider.

> **Note**
>
> To improve federation resiliency, we recommend that you configure your IdP and AWS federation to support multiple SAML sign-in endpoints. For details, see the AWS Security Blog article How to use regional SAML endpoints for failover.

Configure your organization's IdP and AWS to trust each other

1. Register AWS as a service provider (SP) with the IdP of your organization. Use the SAML metadata document from `https://region-code.signin.aws.amazon.com/static/saml-metadata.xml`

   For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints.

   You can optionally use the SAML metadata document from `https://signin.aws.amazon.com/static/saml-metadata.xml`.
2. Using your organization's IdP, you generate an equivalent SAML metadata XML file that can describe your IdP as an IAM identity provider in AWS. It must include the issuer name, a creation date, an expiration date, and keys that AWS can use to validate authentication responses (assertions) from your organization.

   If you allow encrypted SAML assertions to be sent from your external IdP, you must generate a private key file using your organization's IdP, and upload this file to your IAM SAML

configuration in .pem file format. AWS STS needs this private key to decrypt SAML responses that correspond to the public key uploaded to your IdP.

> **Note**
>
> As defined by the SAML V2.0 Metadata Interoperability Profile Version 1.0, IAM does not evaluate or take action on the expiration of X.509 certificates in SAML metadata documents. If you are concerned about expired X.509 certificates, we recommend monitoring certificate expiration dates and rotating certificates according to your organization's governance and security policies.

3. In the IAM console, you create a SAML identity provider. As part of this process, you upload the SAML metadata document and private decryption key that was produced by the IdP in your organization in Step 2. For more information, see Create a SAML identity provider in IAM.
4. In IAM, you create one or more IAM roles. In the role's trust policy, you set the SAML provider as the principal, which establishes a trust relationship between your organization and AWS. The role's permission policy establishes what users from your organization are allowed to do in AWS. For more information, see Create a role for a third-party identity provider.

   > **Note**
   >
   > SAML IDPs used in a role trust policy must be in the same account that the role is in.

5. In your organization's IdP, you define assertions that map users or groups in your organization to the IAM roles. Note that different users and groups in your organization might map to different IAM roles. The exact steps for performing the mapping depend on what IdP you're using. In the earlier scenario of an Amazon S3 folder for users, it's possible that all users will map to the same role that provides Amazon S3 permissions. For more information, see Configure SAML assertions for the authentication response.

   If your IdP enables SSO to the AWS console, then you can configure the maximum duration of the console sessions. For more information, see Enabling SAML 2.0 federated principals to access the AWS Management Console.
6. In the application that you're creating, you call the AWS Security Token Service `AssumeRoleWithSAML` API, passing it the ARN of the SAML provider you created in Step 3, the ARN of the role to assume that you created in Step 4, and the SAML assertion about the

current user that you get from your IdP. AWS makes sure that the request to assume the role comes from the IdP referenced in the SAML provider.

For more information, see AssumeRoleWithSAML in the *AWS Security Token Service API Reference*.

7. If the request is successful, the API returns a set of temporary security credentials, which your application can use to make signed requests to AWS. Your application has information about the current user and can access user-specific folders in Amazon S3, as described in the previous scenario.

Overview of the role to allow SAML-federated access to your AWS resources

The roles that you create in IAM define what SAML federated principals from your organization are allowed to do in AWS. When you create the trust policy for the role, you specify the SAML provider that you created earlier as the `Principal`. You can additionally scope the trust policy with a `Condition` to allow only users that match certain SAML attributes to access the role. For example, you can specify that only users whose SAML affiliation is `staff` (as asserted by https://openidp.feide.no) are allowed to access the role, as illustrated by the following sample policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"Federated": "arn:aws:iam::111122223333:saml-provider/
ExampleOrgSSOProvider"},
    "Action": "sts:AssumeRoleWithSAML",
    "Condition": {
      "StringEquals": {
        "saml:aud": "https://us-east-1.signin.aws.amazon.com/saml",
        "saml:iss": "https://openidp.feide.no"
      },
      "ForAllValues:StringLike": {"saml:edupersonaffiliation": ["staff"]}
    }
  }]
}
```

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws-cn:iam::111122223333:saml-provider/
ExampleOrgSSOProvider"
            },
            "Action": "sts:AssumeRoleWithSAML",
            "Condition": {
                "StringEquals": {
                    "saml:aud": "https://cn-north-1.signin.amazonaws.cn/saml",
                    "saml:iss": "https://openidp.feide.no"
                },
                "ForAllValues:StringLike": {
                    "saml:edupersonaffiliation": [
                        "staff"
                    ]
                }
            }
        }
    ]
}
```

Note

SAML IDPs used in a role trust policy must be in the same account that the role is in.

The `saml:aud` context key in the policy specifies the URL your browser displays when signing into the console. This sign-in endpoint URL must match your identity provider's recipient attribute. You can include sign-in URLs within particular regions. AWS recommends using Regional endpoints instead of the global endpoint to improve federation resiliency. If you have only one endpoint configured, you won't be able to federate into AWS in the unlikely event that the endpoint becomes unavailable. For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints.

The following example shows the sign-in URL format with the optional `region-code`.

`https://region-code.signin.aws.amazon.com/saml`

If SAML encryption is required, the sign-in URL must include the unique identifier AWS assigns to your SAML provider, which you can find on the Identity provider detail page. In the following example, the sign-in URL includes the IdP unique identifier, which requires /acs/ be appended to the sign-in path.

`https://region-code.signin.aws.amazon.com/saml/acs/IdP-ID`

For the permission policy in the role, you specify permissions as you would for any role. For example, if users from your organization are allowed to administer Amazon Elastic Compute Cloud instances, you must explicitly allow Amazon EC2 actions in the permissions policy, such as those in the **AmazonEC2FullAccess** managed policy.

For more information about the SAML keys that you can check in a policy, see Available keys for SAML-based AWS STS federation.

Uniquely identifying users in SAML-based federation

When you create access policies in IAM, it's often useful to be able to specify permissions based on the identity of users. For example, for users who have been federated using SAML, an application might want to keep information in Amazon S3 using a structure like this:

```
amzn-s3-demo-bucket/app1/user1
amzn-s3-demo-bucket/app1/user2
amzn-s3-demo-bucket/app1/user3
```

You can create the bucket (`amzn-s3-demo-bucket`) and folder (`app1`) through the Amazon S3 console or the AWS CLI, since those are static values. However, the user-specific folders (*`user1`*, *`user2`*, *`user3`*, etc.) have to be created at run time using code, since the value that identifies the user isn't known until the first time the user signs in through the federation process.

To write policies that reference user-specific details as part of a resource name, the user identity has to be available in SAML keys that can be used in policy conditions. The following keys are available for SAML 2.0–based federation for use in IAM policies. You can use the values returned by the following keys to create unique user identifiers for resources like Amazon S3 folders.

- `saml:namequalifier`. A hash value based on the concatenation of the `Issuer` response value (`saml:iss`) and a string with the AWS account ID and the friendly name (the last part of the

ARN) of the SAML provider in IAM. The concatenation of the account ID and friendly name of the SAML provider is available to IAM policies as the key `saml:doc`. The account ID and provider name must be separated by a '/' as in "123456789012/provider_name". For more information, see the `saml:doc` key at Available keys for SAML-based AWS STS federation.

The combination of `NameQualifier` and `Subject` can be used to uniquely identify a SAML federated principal. The following pseudocode shows how this value is calculated. In this pseudocode + indicates concatenation, `SHA1` represents a function that produces a message digest using SHA-1, and `Base64` represents a function that produces Base-64 encoded version of the hash output.

```
Base64 ( SHA1 ( "https://example.com/saml" + "123456789012" + "/
MySAMLIdP" ) )
```

For more information about the policy keys that are available for SAML-based federation, see Available keys for SAML-based AWS STS federation.

- `saml:sub` (string). This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).
- `saml:sub_type` (string). This key can be `persistent`, `transient`, or the full `Format` URI from the `Subject` and `NameID` elements used in your SAML assertion. A value of `persistent` indicates that the value in `saml:sub` is the same for a user across all sessions. If the value is `transient`, the user has a different `saml:sub` value for each session. For information about the `NameID` element's `Format` attribute, see Configure SAML assertions for the authentication response.

The following example shows a permission policy that uses the preceding keys to grant permissions to a user-specific folder in Amazon S3. The policy assumes that the Amazon S3 objects are identified using a prefix that includes both `saml:namequalifier` and `saml:sub`. Notice that the `Condition` element includes a test to be sure that `saml:sub_type` is set to `persistent`. If it is set to `transient`, the `saml:sub` value for the user can be different for each session, and the combination of values should not be used to identify user-specific folders.

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:PutObject",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket-org-data/backup/${saml:namequalifier}/
${saml:sub}",
      "arn:aws:s3:::amzn-s3-demo-bucket-org-data/backup/${saml:namequalifier}/
${saml:sub}/*"
    ],
    "Condition": {"StringEquals": {"saml:sub_type": "persistent"}}
  }
}
```

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:PutObject",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws-cn:s3:::amzn-s3-demo-bucket-org-data/backup/${saml:namequalifier}/
${saml:sub}",
      "arn:aws-cn:s3:::amzn-s3-demo-bucket-org-data/backup/${saml:namequalifier}/
${saml:sub}/*"
    ],
    "Condition": {"StringEquals": {"saml:sub_type": "persistent"}}
  }
}
```

For more information about mapping assertions from the IdP to policy keys, see Configure SAML assertions for the authentication response.

Create a SAML identity provider in IAM

An IAM SAML 2.0 identity provider is an entity in IAM that describes an external identity provider (IdP) service that supports the SAML 2.0 (Security Assertion Markup Language 2.0) standard. You use an IAM identity provider when you want to establish trust between a SAML-compatible IdP such as Shibboleth or Active Directory Federation Services and AWS, so that your users can access AWS resources. IAM SAML identity providers are used as principals in an IAM trust policy.

For more information about this scenario, see SAML 2.0 federation.

You can create and manage an IAM identity provider in the AWS Management Console or with AWS CLI, Tools for Windows PowerShell, or AWS API calls.

After you create a SAML provider, you must create one or more IAM roles. A role is an identity in AWS that doesn't have its own credentials (as a user does). But in this context, a role is dynamically assigned to a SAML federated principal that is authenticated by your IdP. The role permits your IdP to request temporary security credentials for access to AWS. The policies assigned to the role determine what users are allowed to do in AWS. To create a role for SAML federation, see Create a role for a third-party identity provider.

Finally, after you create the role, you complete the SAML trust by configuring your IdP with information about AWS and the roles that you want your SAML federated principals to use. This is referred to as configuring relying party trust between your IdP and AWS. To configure relying party trust, see Configure your SAML 2.0 IdP with relying party trust and adding claims.

Topics

- Prerequisites
- Create and manage an IAM SAML identity provider (console)
- Manage SAML encryption keys
- Create and manage an IAM SAML Identity Provider (AWS CLI)
- Create and manage an IAM SAML identity provider (AWS API)
- Next steps

Prerequisites

Before you can create a SAML identity provider, you must have the following information from your IdP.

- Get the SAML metadata document from your IdP. This document includes the issuer's name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) that are received from the IdP. To generate the metadata document, use the identity management software provided by your external IdP.

  > **Important**
  >
  > This metadata file includes the issuer name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) received from the IdP. The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.
  >
  > The X.509 certificate included as part of the SAML metadata document must use a key size of at least 1024 bits. Also, the X.509 certificate must also be free of any repeated extensions. You can use extensions, but the extensions can only appear once in the certificate. If the X.509 certificate does not meet either condition, IdP creation fails and returns an "Unable to parse metadata" error.
  >
  > As defined by the SAML V2.0 Metadata Interoperability Profile Version 1.0, IAM does not evaluate or take action on the expiration of X.509 certificates in SAML metadata documents. If you are concerned about expired X.509 certificates, we recommend monitoring certificate expiration dates and rotating certificates according to your organization's governance and security policies.

- When you choose to enable SAML encryption, you must generate a private key file using your IdP, and upload this file to your IAM SAML configuration in .pem file format. AWS STS needs this private key to decrypt SAML responses that correspond to the public key uploaded to your IdP. The following algorithms are supported:
  - Encryption algorithms
    - AES-128
    - AES-256
    - RSA-OAEP
  - Key transport algorithms
    - AES-CBC
    - AES-GCM

  See your identity provider's documentation for steps to generate a private key.

> **Note**
>
> IAM Identity Center and Amazon Cognito do not support encrypted SAML assertions from IAM SAML identity providers. You can indirectly add support for encrypted SAML assertions to Amazon Cognito identity pool federation with Amazon Cognito user pools. User pools have SAML federation that's independent of IAM SAML federation and supports SAML signing and encryption. Although this feature doesn't extend directly to identity pools, user pools can be IdPs to identity pools. To use SAML encryption with identity pools, add a SAML provider with encryption to a user pool that is an IdP to an identity pool.
> Your SAML provider must be able to encrypt SAML assertions with a key that your user pool provides. User pools won't accept assertions encrypted with a certificate that IAM has provided.

For instructions on how to configure many of the available IdPs to work with AWS, including how to generate the required SAML metadata document, see Integrate third-party SAML solution providers with AWS.

For help with SAML federation, see Troubleshooting SAML federation.

Create and manage an IAM SAML identity provider (console)

You can use the AWS Management Console to create, update, and delete IAM SAML identity providers. For help with SAML federation, see Troubleshooting SAML federation.

To create an IAM SAML identity provider (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers** and then choose **Add provider**.
3. For **Configure provider**, choose **SAML**.
4. Type a name for the identity provider.
5. For **Metadata document**, choose **Choose file**, specify the SAML metadata document that you downloaded in the section called “Prerequisites”.

> **Note**
>
> The **Valid until** date displayed for the identity provider is set by IAM at provider creation time. It does not reflect the `validUntil` or `cacheDuration` attributes in your SAML metadata document. IAM does not use these metadata attributes to determine the **Valid until** date.
> IAM does not evaluate or take action on the expiration of the SAML metadata document or X.509 certificates in SAML metadata documents during SAML authentication. SAML authentication validity is enforced at the assertion level, using the `NotOnOrAfter` attribute in the assertion's `Conditions` and `SubjectConfirmationData` elements. If you are concerned about expired X.509 certificates, we recommend monitoring certificate expiration dates and rotating certificates according to your organization's governance and security policies.

6. (Optional) For **SAML encryption**, choose **Choose file** and select the private key file that you created in the section called "Prerequisites". Choose **Require encryption** to accept only encrypted requests from your IdP.
7. (Optional) For **Add tags**, you can add key–value pairs to help you identify and organize your IdPs. You can also use tags to control access to AWS resources. To learn more about tagging SAML identity providers, see Tag IAM SAML identity providers.

   Choose **Add tag**. Enter values for each tag key-value pair.
8. Verify the information that you have provided. When you are done, choose **Add provider**.
9. Assign an IAM role to your identity provider. This role gives external user identities managed by your identity provider permissions to access AWS resources in your account. To learn more about creating roles for identity federation, see Create a role for a third-party identity provider.

> **Note**
>
> SAML IDPs used in a role trust policy must be in the same account that the role is in.

To delete a SAML provider (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers**.
3. Select the radio button next to the identity provider that you want to delete.
4. Choose **Delete**. A new window opens.
5. Confirm that you want to delete the provider by typing the word `delete` in the field. Then, choose **Delete**.

Manage SAML encryption keys

You can configure IAM SAML providers to receive encrypted assertions in the SAML response from your external IdP. Users can assume a role in AWS with encrypted SAML assertions by calling `sts:AssumeRoleWithSAML`.

SAML encryption ensures that assertions are secure when passed through intermediaries or third parties. In addition, this feature helps you meet FedRAMP or any internal compliance policy requirements that mandate SAML assertions to be encrypted.

To configure an IAM SAML identity provider, see Create a SAML identity provider in IAM. For help with SAML federation, see Troubleshooting SAML federation.

Rotate SAML encryption key

IAM uses the private key you uploaded to the IAM SAML provider to decrypt encrypted SAML assertions from your IdP. You can save up to two private key files for each identity provider, allowing you to rotate private keys as necessary. When two files are saved, each request will first attempt to decrypt with the newest **Added on** date, then IAM attempts to decrypt the request with the oldest **Added on** date.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Identity providers** and then select your provider from the list.
3. Choose the **SAML encryption** tab and choose **Add new key**.
4. Select **Choose file** and upload the private key you downloaded from your IdP as a .pem file.Then, choose **Add key**.

5. In the **Private keys for SAML decryption** section, select the expired private key file and choose **Remove**. We recommend you remove the expired private key after adding a new private key to ensure the first attempt to decrypt your assertion is successful.

Create and manage an IAM SAML Identity Provider (AWS CLI)

You can use the AWS CLI to create, update, and delete SAML providers. For help with SAML federation, see Troubleshooting SAML federation.

To create an IAM identity provider and upload a metadata document (AWS CLI)

- Run this command: `aws iam create-saml-provider`

To update an IAM SAML identity provider (AWS CLI)

You can update the metadata file, SAML encryption settings, and rotate private key decryption files for your IAM SAML provider. To rotate private keys, add your new private key and then remove the old key in a separate request. For more information about rotating private keys, see Manage SAML encryption keys.

- Run this command:`aws iam update-saml-provider`

To tag an existing IAM identity provider (AWS CLI)

- Run this command:`aws iam tag-saml-provider`

To list tags for existing IAM identity provider (AWS CLI)

- Run this command:`aws iam list-saml-provider-tags`

To remove tags on an existing IAM identity provider (AWS CLI)

- Run this command:`aws iam untag-saml-provider`

To delete an IAM SAML identity provider (AWS CLI)

1. (Optional) To list information for all providers, such as the ARN, creation date, and expiration, run the following command:

- `aws iam list-saml-providers`

2. (Optional) To get information about a specific provider, such as the ARN, creation date, expiration date, encryption settings, and private key information, run the following command:

   - `aws iam get-saml-provider`

3. To delete an IAM identity provider, run the following command:

   - `aws iam delete-saml-provider`

Create and manage an IAM SAML identity provider (AWS API)

You can use the AWS API to create, update, and delete SAML providers. For help with SAML federation, see Troubleshooting SAML federation.

To create an IAM identity provider and upload a metadata document (AWS API)

- Call this operation: `CreateSAMLProvider`

To update an IAM SAML identity provider (AWS API)

You can update the metadata file, SAML encryption settings, and rotate private key decryption files for your IAM SAML provider. To rotate private keys, add your new private key and then remove the old key in a separate request. For more information about rotating private keys, see Manage SAML encryption keys.

- Call this operation: `UpdateSAMLProvider`

To tag an existing IAM identity provider (AWS API)

- Call this operation: `TagSAMLProvider`

To list tags for an existing IAM identity provider (AWS API)

- Call this operation: `ListSAMLProviderTags`

To remove tags on an existing IAM identity provider (AWS API)

- Call this operation: `UntagSAMLProvider`

To delete an IAM identity provider (AWS API)

1. (Optional) To list information for all IdPs, such as the ARN, creation date, and expiration, call the following operation:
   - `ListSAMLProviders`
2. (Optional) To get information about a specific provider, such as the ARN, creation date, expiration date, encryption settings, and private key information, call the following operation:
   - `GetSAMLProvider`
3. To delete an IdP, call the following operation:
   - `DeleteSAMLProvider`

Next steps

After you create a SAML identity provider, set up the relying party trust with your IdP. You can also use claims from your IdP's authentication response in policies to control access to a role.

- You must tell the IdP about AWS as a service provider. This is called adding relying party trust between your IdP and AWS. The exact process for adding relying party trust depends on what IdP you're using. For details, see Configure your SAML 2.0 IdP with relying party trust and adding claims.
- When the IdP sends the response containing the claims to AWS, many of the incoming claims map to AWS context keys. You can use these context keys in IAM policies using the Condition element to control access to a role. For details, see Configure SAML assertions for the authentication response

Configure your SAML 2.0 IdP with relying party trust and adding claims

When you create an IAM identity provider and role for SAML access, you are telling AWS about the external identity provider (IdP) and what its users are allowed to do. Your next step is to then tell the IdP about AWS as a service provider. This is called adding *relying party trust* between your IdP

and AWS. The exact process for adding relying party trust depends on what IdP you're using. For details, see the documentation for your identity management software.

Many IdPs allow you to specify a URL from which the IdP can read an XML document that contains relying party information and certificates. For AWS, use the sign-in endpoint URL. The following example shows the URL format with the optional `region-code`.

`https://region-code.signin.aws.amazon.com/static/saml-metadata.xml`

If SAML encryption is required, the URL must include the unique identifier AWS assigns to your SAML provider, which you can find on the Identity provider detail page. The following example shows a regional sign-in URL that includes a unique identifier.

`https://region-code.signin.aws.amazon.com/static/saml/IdP-ID/saml-metadata.xml`

For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints. For the AWS value, you can also use the non-Regional endpoint `https://signin.aws.amazon.com/saml`.

If you can't specify a URL directly, then download the XML document from the preceding URL and import it into your IdP software.

You also need to create appropriate claim rules in your IdP that specify AWS as a relying party. When the IdP sends a SAML response to the AWS endpoint, it includes a SAML *assertion* that contains one or more *claims*. A claim is information about the user and its groups. A claim rule maps that information into SAML attributes. This lets you make sure that SAML authentication responses from your IdP contain the necessary attributes that AWS uses in IAM policies to check permissions for SAML federated principals. For more information, see the following topics:

- Overview of the role to allow SAML-federated access to your AWS resources. This topic discusses using SAML-specific keys in IAM policies and how to use them to restrict permissions for SAML federated principals.
- Configure SAML assertions for the authentication response. This topic discusses how to configure SAML claims that include information about the user. The claims are bundled into a SAML assertion and included in the SAML response that is sent to AWS. You must ensure that the information needed by AWS policies is included in the SAML assertion in a form that AWS can recognize and use.

- Integrate third-party SAML solution providers with AWS. This topic provides links to documentation provided by third-party organizations about how to integrate identity solutions with AWS.

> **Note**
>
> To improve federation resiliency, we recommend that you configure your IdP and AWS federation to support multiple SAML sign-in endpoints. For details, see the AWS Security Blog article How to use regional SAML endpoints for failover.

Integrate third-party SAML solution providers with AWS

> **Note**
>
> We recommend that you require your human users to use temporary credentials when accessing AWS. Have you considered using AWS IAM Identity Center? You can use IAM Identity Center to centrally manage access to multiple AWS accounts and provide users with MFA-protected, single sign-on access to all their assigned accounts from one place. With IAM Identity Center, you can create and manage user identities in IAM Identity Center or easily connect to your existing SAML 2.0 compatible identity provider. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

The following links help you configure third-party SAML 2.0 identity provider (IdP) solutions to work with AWS federation. Check with your identity provider to determine whether they support SAML token encryption. For SAML encryption requirements, see Manage SAML encryption keys.

> **Tip**
>
> AWS Support engineers can assist customers who have business and enterprise support plans with some integration tasks that involve third-party software. For a current list of supported platforms and applications, see What third-party software is supported? in the *AWS Support FAQs*.

Solution	More information
Auth0	Integrate with Amazon Web Services – This page on the Auth0 documentation website has links to resources that describe how to set up single sign-on (SSO) with the AWS Management Console and includes a JavaScript example. You can configure Auth0 to pass session tags. For more information, see Auth0 Announces Partnership with AWS for IAM Session Tags.
Microsoft Entra	Tutorial: Microsoft Entra SSO integration with AWS Single-Account Access – This tutorial on the Microsoft website describes how to set up Microsoft Entra (formerly known as Azure AD) as an identity provider (IdP) using SAML federation.
Centrify	Configure Centrify and Use SAML for SSO to AWS – This page on the Centrify website explains how to configure Centrify to use SAML for SSO to AWS.
CyberArk	Configure CyberArk to provide Amazon Web Services (AWS) access to users logging in through SAML single sign-on (SSO) from the CyberArk User Portal.
ForgeRock	The ForgeRock Identity Platform integrates with AWS. You can configure ForgeRock to pass session tags. For more information, see Attribute Based Access Control for Amazon Web Services.
Google Workspace	Amazon Web Services cloud application – This article on the Google Workspace Admin Help site describes how to configure Google Workspace as a SAML 2.0 IdP with AWS as the service provider.
IBM	You can configure IBM to pass session tags. For more information, see IBM Cloud Identity IDaaS one of first to support AWS session tags.

Solution	More information
JumpCloud	Granting Access via IAM Roles for Single Sign On (SSO) with Amazon AWS – This article on the JumpCloud website describes how to set up and enable SSO based on IAM roles for AWS.
Matrix42	MyWorkspace Getting Started Guide – This guide describes how to integrate AWS identity services with Matrix42 MyWorkspace.
Microsoft Active Directory Federation Services (AD FS)	Field Notes: Integrating Active Directory Federation Service with AWS IAM Identity Center – This post on the AWS Architecture Blog explains the authentication flow between AD FS and AWS IAM Identity Center (IAM Identity Center). IAM Identity Center supports identity federation with SAML 2.0, allowing integration with AD FS solutions. Users can sign in to the IAM Identity Center portal with their corporate credentials reducing the admin overhead of maintaining separate credentials on IAM Identity Center. You can also configure AD FS to pass session tags. For more information, see Use attribute-based access control with AD FS to simplify IAM permissions management.
miniOrange	SSO for AWS – This page on the miniOrange website describes how to establish secure access to AWS for enterprises and full control over access of AWS applications.
Okta	Integrating the Amazon Web Services Command Line Interface Using Okta – From this page on the Okta support site you can learn how to configure Okta for use with AWS. You can configure Okta to pass session tags. For more information, see Okta and AWS Partner to Simplify Access Via Session Tags.

Solution	More information
Okta	AWS Account Federation – This section on the Okta website describes how to set up and enable IAM Identity Center for AWS.
OneLogin	From the OneLogin Knowledgebase, search for **SAML AWS** for a list of articles that explain how to set up IAM Identity Center functionality between OneLogin and AWS for a single-role and multi-role scenarios. You can configure OneLogin to pass session tags. For more information, see OneLogin and Session Tags: Attribute-Based Access Control for AWS Resources.
Ping Identity	PingFederate AWS Connector – View details about the PingFederate AWS Connector, a quick connectio n template to easily set up a single sign-on (SSO) and provisioning connection. Read documentation and download the latest PingFederate AWS Connector for integrations with AWS. You can configure Ping Identity to pass session tags. For more information, see Announcing Ping Identity Support for Attribute-Based Access Control in AWS.
RadiantLogic	Radiant Logic Technology Partners – Radiant Logic's RadiantOne Federated Identity Service integrates with AWS to provide an identity hub for SAML-based SSO.
RSA	Amazon Web Services - RSA Ready Implementation Guide provides guidance for integrating AWS and RSA. For more information on SAML configuration, see Amazon Web Services - SAML My Page SSO Configuration - RSA Ready Implementation Guide.
Salesforce.com	How to configure SSO from Salesforce to AWS – This how-to article on the Salesforce.com developer site describes how to set up an identity provider (IdP) in Salesforce and configure AWS as a service provider.

Solution	More information
SecureAuth	AWS - SecureAuth SAML SSO – This article on the SecureAuth website describes how to set up SAML integration with AWS for a SecureAuth appliance.
Shibboleth	How to Use Shibboleth for SSO to the AWS Management Console – This entry on the AWS Security Blog provides a step-by-step tutorial on how to set up Shibboleth and configure it as an identity provider for AWS. You can configure Shibboleth to pass session tags.

For more details, see the IAM Partners page on the AWS website.

Configure SAML assertions for the authentication response

After you have verified a user's identity in your organization, the external identity provider (IdP) sends an authentication response to the AWS sign-in endpoint URL. This response is a POST request that includes a SAML token that adheres to the HTTP POST Binding for SAML 2.0 standard and that contains the following elements, or *claims*. You configure these claims in your SAML-compatible IdP. Refer to the documentation for your IdP for instructions on how to enter these claims.

When the IdP sends the response containing the claims to AWS, many of the incoming claims map to AWS context keys. These context keys can be checked in IAM policies using the `Condition` element. A listing of the available mappings follows in the section Mapping SAML attributes to AWS trust policy context keys.

`Subject` and `NameID`

The response must include exactly one `SubjectConfirmation` element with a `SubjectConfirmationData` element that includes both the `NotOnOrAfter` attribute and a `Recipient` attribute. The Recipient attribute must include a value that matches the AWS sign-in endpoint URL. Your IdP may use the term ACS, `Recipient`, or `Target` to refer to this attribute.

If SAML encryption is required, the sign-in URL must include the unique identifier AWS assigns to your SAML provider, which you can find on the Identity provider detail page. The following example shows the sign-in URL format with the optional `region-code`.

`https://region-code.signin.aws.amazon.com/saml`

In the following example, the sign-in URL includes a unique identifier, which requires /acs/ be appended to the sign-in path.

`https://region-code.signin.aws.amazon.com/saml/acs/IdP-ID`

For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints. For the AWS value, you can also use the global sign-in endpoint `https://signin.aws.amazon.com/saml`.

`NameID` elements can have the value persistent, transient, or consist of the full Format URI as provided by the IdP solution. A value of persistent indicates that the value in `NameID` is the same for a user between sessions. If the value is transient, the user has a different `NameID` value for each session. Single sign-on interactions support the following types of identifiers:

- `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName`
- `urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`

The following excerpt shows an example. Substitute your own values for the marked ones.

```
<Subject>
  <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:persistent">_cbb88bf52c2510eabe00c1642d4643f41430fe25e3</NameID>
  <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <SubjectConfirmationData NotOnOrAfter="2013-11-05T02:06:42.876Z"
 Recipient="https://region-code.signin.aws.amazon.com/saml/SAMLSP4SHN3UIS2D558H46"/>
  </SubjectConfirmation>
</Subject>
```

> **Important**
>
> The `saml:aud` context key comes from the SAML *recipient* attribute because it is the SAML equivalent to the OIDC audience field, for example, `accounts.google.com:aud`.

`PrincipalTag` SAML attribute

(Optional) You can use an `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/PrincipalTag:{TagKey}`. This element allows you to pass attributes as session tags in the SAML assertion. For more information about session tags, see Pass session tags in AWS STS.

To pass attributes as session tags, include the `AttributeValue` element that specifies the value of the tag. For example, to pass the tag key-value pairs `Project` = `Marketing` and `CostCenter` = `12345`, use the following attribute. Include a separate `Attribute` element for each tag.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:Project">
  <AttributeValue>Marketing</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:CostCenter">
  <AttributeValue>12345</AttributeValue>
</Attribute>
```

To set the tags above as transitive, include another `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys`. This is an optional multivalued attribute that sets your session tags as transitive. Transitive tags persist when you use the SAML session to assume another role in AWS. This is known as role chaining. For example, to set both the `Principal` and `CostCenter` tags as transitive, use the following attribute to specify the keys.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys">
  <AttributeValue>Project</AttributeValue>
  <AttributeValue>CostCenter</AttributeValue>
</Attribute>
```

`Role` SAML attribute

You can use an `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/Role`. This element contains one or more `AttributeValue` elements that

list the IAM identity provider and role to which the user is mapped by your IdP. The IAM role and IAM identity provider are specified as a comma-delimited pair of ARNs in the same format as the `RoleArn` and `PrincipalArn` parameters that are passed to AssumeRoleWithSAML. This element must contain at least one role-provider pair (`AttributeValue` element), and can contain multiple pairs. If the element contains multiple pairs, then the user is asked to choose which role to assume when they use WebSSO to sign in to the AWS Management Console.

> **Important**
>
> The value of the `Name` attribute in the `Attribute` tag is case-sensitive. It must be set to `https://aws.amazon.com/SAML/Attributes/Role` exactly.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/Role">
  <AttributeValue>arn:aws:iam::account-number:role/role-name1,arn:aws:iam::account-
number:saml-provider/provider-name</AttributeValue>
  <AttributeValue>arn:aws:iam::account-number:role/role-name2,arn:aws:iam::account-
number:saml-provider/provider-name</AttributeValue>
  <AttributeValue>arn:aws:iam::account-number:role/role-name3,arn:aws:iam::account-
number:saml-provider/provider-name</AttributeValue>
</Attribute>
```

`RoleSessionName` SAML attribute

You can use an `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/RoleSessionName`. This element contains one `AttributeValue` element that provides an identifier for the temporary credentials that are issued when the role is assumed. You can use this to associate the temporary credentials with the user who is using your application. This element is used to display user information in the AWS Management Console. The value in the `AttributeValue` element must be between 2 and 64 characters long, can contain only alphanumeric characters, underscores, and the following characters: **. , + = @ -** (hyphen). It cannot contain spaces. The value is typically a user ID (`john`) or an email address (`johndoe@example.com`). It should not be a value that includes a space, like a user's display name (`John Doe`).

> **Important**
>
> The value of the `Name` attribute in the `Attribute` tag is case-sensitive. It must be set to `https://aws.amazon.com/SAML/Attributes/RoleSessionName` exactly.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/RoleSessionName">
  <AttributeValue>user-id-name</AttributeValue>
</Attribute>
```

`SessionDuration` SAML attribute

(Optional) You can use an `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/SessionDuration`. This element contains one `AttributeValue` element that specifies how long the user can access the AWS Management Console before having to request new temporary credentials. The value is an integer representing the number of seconds for the session. The value can range from 900 seconds (15 minutes) to 43200 seconds (12 hours). If this attribute is not present, then the credential last for one hour (the default value of the `DurationSeconds` parameter of the `AssumeRoleWithSAML` API).

To use this attribute, you must configure the SAML provider to provide single sign-on access to the AWS Management Console through the console sign-in web endpoint at `https://region-code.signin.aws.amazon.com/saml`. For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints. You can optionally use the following URL: `https://signin.aws.amazon.com/static/saml`. Note that this attribute extends sessions only to the AWS Management Console. It cannot extend the lifetime of other credentials. However, if it is present in an `AssumeRoleWithSAML` API call, it can be used to *shorten* the duration of the session. The default lifetime of the credentials returned by the call is 60 minutes.

Note, too, that if a `SessionNotOnOrAfter` attribute is also defined, then the ***lesser*** value of the two attributes, `SessionDuration` or `SessionNotOnOrAfter`, establishes the maximum duration of the console session.

When you enable console sessions with an extended duration the risk of compromise of the credentials rises. To help you mitigate this risk, you can immediately disable the active console sessions for any role by choosing **Revoke Sessions** on the **Role Summary** page in the IAM console. For more information, see Revoke IAM role temporary security credentials.

Important

The value of the `Name` attribute in the `Attribute` tag is case-sensitive. It must be set to `https://aws.amazon.com/SAML/Attributes/SessionDuration` exactly.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/SessionDuration">
  <AttributeValue>1800</AttributeValue>
</Attribute>
```

`SourceIdentity` SAML attribute

(Optional) You can use an `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/SourceIdentity`. This element contains one `AttributeValue` element that provides an identifier for the person or application that is using an IAM role. The value for source identity persists when you use the SAML session to assume another role in AWS known as role chaining. The value for source identity is present in the request for every action taken during the role session. The value that is set cannot be changed during the role session. Administrators can then use AWS CloudTrail logs to monitor and audit the source identity information to determine who performed actions with shared roles.

The value in the `AttributeValue` element must be between 2 and 256 characters long, can contain only alphanumeric characters, underscores, and the following characters: **. , + = @ -** (hyphen). It cannot contain spaces. The value is typically an attribute that is associated with the user such as a user id (`john`) or an email address (`johndoe@example.com`). It should not be a value that includes a space, like a user's display name (`John Doe`). For more information about using source identity, see Monitor and control actions taken with assumed roles.

Important

If your SAML assertion is configured to use the `SourceIdentity` attribute, then your role trust policy must also include the `sts:SetSourceIdentity` action, otherwise the assume role operation will fail. For more information about using source identity, see Monitor and control actions taken with assumed roles.

To pass a source identity attribute, include the `AttributeValue` element that specifies the value of the source identity. For example, to pass the source identity `Diego` use the following attribute.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/SourceIdentity">
  <AttributeValue>Diego</AttributeValue>
```

Mapping SAML attributes to AWS trust policy context keys

The tables in this section list commonly used SAML attributes and how they map to trust policy condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the keys to the values that are included in the assertions that accompany a SAML access request.

> **Important**
>
> These keys are available only in IAM trust policies (policies that determine who can assume a role) and are not applicable to permissions policies.

In the eduPerson and eduOrg attributes table, values are typed either as strings or as lists of strings. For string values, you can test these values in IAM trust policies using `StringEquals` or `StringLike` conditions. For values that contain a list of strings, you can use the `ForAnyValue` and `ForAllValues` policy set operators to test the values in trust policies.

> **Note**
>
> You should include only one claim per AWS context key. If you include more than one, only one claim will be mapped.

The following table shows eduPerson and eduOrg attributes.

eduPerson or eduOrg attribute (`Name` key)	Maps to this AWS context key (`FriendlyName` key)	Type
`urn:oid:1.3.6.1.4.1.5923.1.1.1.1`	`eduPerson Affiliation`	List of strings
`urn:oid:1.3.6.1.4.1.5923.1.1.1.2`	`eduPersonNickname`	List of strings

eduPerson or eduOrg attribute (Name key)	Maps to this AWS context key (FriendlyName key)	Type
urn:oid:1.3.6.1.4.1.5923.1.1.1.3	eduPersonOrgDN	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.4	eduPerson OrgUnitDN	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.5	eduPerson PrimaryAf filiation	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.6	eduPerson PrincipalName	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.7	eduPerson Entitlement	List of strings
urn:oid:1.3.6.1.4.1.5923.1.1.1.8	eduPerson PrimaryOrgUnitDN	String
urn:oid:1.3.6.1.4.1.5923.1.1.1.9	eduPerson ScopedAff iliation	List of strings
urn:oid:1.3.6.1.4.1.5923.1. 1.1.10	eduPerson TargetedID	List of strings
urn:oid:1.3.6.1.4.1.5923.1. 1.1.11	eduPerson Assurance	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.2	eduOrgHomePageURI	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.3	eduOrgIde ntityAuth NPolicyURI	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.4	eduOrgLegalName	List of strings

eduPerson or eduOrg attribute (Name key)	Maps to this AWS context key (FriendlyName key)	Type
urn:oid:1.3.6.1.4.1.5923.1.2.1.5	eduOrgSuperiorURI	List of strings
urn:oid:1.3.6.1.4.1.5923.1.2.1.6	eduOrgWhitePagesURI	List of strings
urn:oid:2.5.4.3	cn	List of strings

The following table shows Active Directory attributes.

AD attribute	Maps to this AWS context key	Type
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name	name	String
http://schemas.xmlsoap.org/claims/CommonName	commonName	String
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname	givenName	String
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname	surname	String
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress	mail	String
http://schemas.microsoft.com/ws/2008/06/identity/claims/primarygroupsid	uid	String

The following table shows X.500 attributes.

X.500 attribute	Maps to this AWS context key	Type
2.5.4.3	commonName	String
2.5.4.4	surname	String
2.4.5.42	givenName	String
2.5.4.45	x500UniqueIdentifier	String
0.9.2342.19200300100.1.1	uid	String
0.9.2342.19200300100.1.3	mail	String
0.9.2342.19200300.100.1.45	organizationStatus	String

Enabling SAML 2.0 federated principals to access the AWS Management Console

You can use a role to configure your SAML 2.0-compliant identity provider (IdP) and AWS to permit SAML federated principals to access the AWS Management Console. The role grants the user permissions to carry out tasks in the console. If you want to give SAML federated principals other ways to access AWS, see one of these topics:

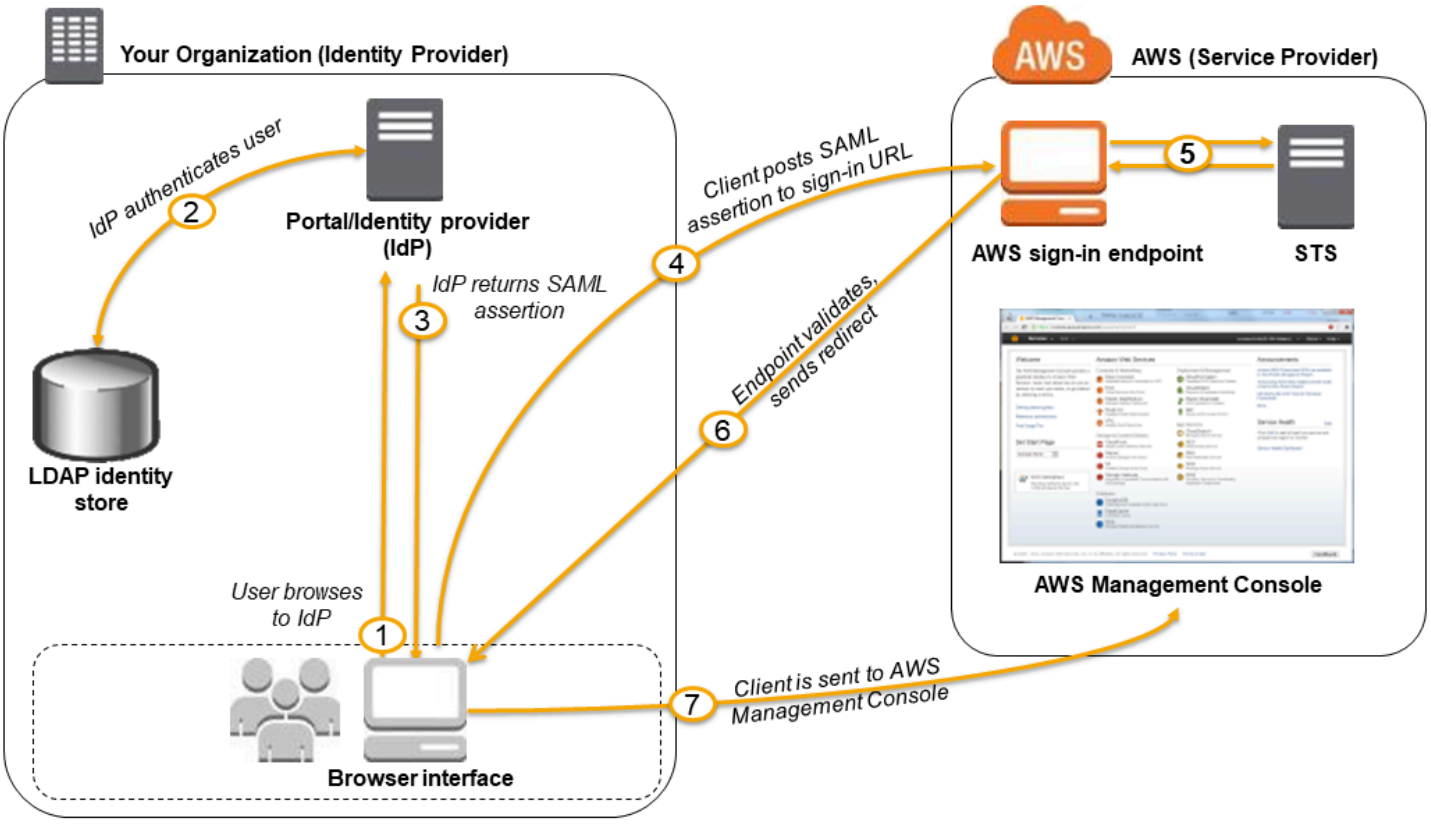
- AWS CLI: Switch to an IAM role (AWS CLI)
- Tools for Windows PowerShell: Switch to an IAM role (Tools for Windows PowerShell)
- AWS API: Switch to an IAM role (AWS API)

Overview

The following diagram illustrates the flow for SAML-enabled single sign-on.

> **Note**
>
> This specific use of SAML differs from the more general one illustrated at SAML 2.0 federation because this workflow opens the AWS Management Console on behalf of the user. This requires the use of the AWS sign-in endpoint instead of directly calling the

`AssumeRoleWithSAML` API. The endpoint calls the API for the user and returns a URL that automatically redirects the user's browser to the AWS Management Console.



The diagram illustrates the following steps:

1. The user browses to your organization's portal and selects the option to go to the AWS Management Console. In your organization, the portal is typically a function of your IdP that handles the exchange of trust between your organization and AWS. For example, in Active Directory Federation Services, the portal URL is: `https://ADFSServiceName/adfs/ls/IdpInitiatedSignOn.aspx`
2. The portal verifies the user's identity in your organization.
3. The portal generates a SAML authentication response that includes assertions that identify the user and include attributes about the user. You can also configure your IdP to include a SAML assertion attribute called `SessionDuration` that specifies how long the console session is valid. You can also configure the IdP to pass attributes as session tags. The portal sends this response to the client browser.

4. The client browser is redirected to the AWS single sign-on endpoint and posts the SAML assertion.
5. The endpoint requests temporary security credentials on behalf of the user and creates a console sign-in URL that uses those credentials.
6. AWS sends the sign-in URL back to the client as a redirect.
7. The client browser is redirected to the AWS Management Console. If the SAML authentication response includes attributes that map to multiple IAM roles, the user is first prompted to select the role for accessing the console.

From the user's perspective, the process happens transparently: The user starts at your organization's internal portal and ends up at the AWS Management Console, without ever having to supply any AWS credentials.

Consult the following sections for an overview of how to configure this behavior along with links to detailed steps.

Configure your network as a SAML provider for AWS

Inside your organization's network, you configure your identity store (such as Windows Active Directory) to work with a SAML-based IdP like Windows Active Directory Federation Services, Shibboleth, etc. Using your IdP, you generate a metadata document that describes your organization as an IdP and includes authentication keys. You also configure your organization's portal to route user requests for the AWS Management Console to the AWS SAML endpoint for authentication using SAML assertions. How you configure your IdP to produce the metadata.xml file depends on your IdP. Refer to your IdP's documentation for instructions, or see Integrate third-party SAML solution providers with AWS for links to the web documentation for many of the SAML providers supported.

Create a SAML provider in IAM

Next, you sign in to the AWS Management Console and go to the IAM console. There you create a new SAML provider, which is an entity in IAM that holds information about your organization's IdP. As part of this process, you upload the metadata document produced by the IdP software in your organization in the previous section. For details, see Create a SAML identity provider in IAM.

Configure permissions in AWS for SAML federated principals

The next step is to create an IAM role that establishes a trust relationship between IAM and your organization's IdP. This role must identify your IdP as a principal (trusted entity) for purposes of

federation. The role also defines what users authenticated by your organization's IdP are allowed to do in AWS. You can use the IAM console to create this role. When you create the trust policy that indicates who can assume the role, you specify the SAML provider that you created earlier in IAM. You also specify one or more SAML attributes that a user must match to be allowed to assume the role. For example, you can specify that only users whose SAML eduPersonOrgDN value is `ExampleOrg` are allowed to sign in. The role wizard automatically adds a condition to test the `saml:aud` attribute to make sure that the role is assumed only for sign-in to the AWS Management Console.

If SAML encryption is required, the sign-in URL must include the unique identifier AWS assigns to your SAML provider, which you can find on the Identity provider detail page. The trust policy for the role might look like this:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:saml-provider/
ExampleOrgSSOProvider"
            },
            "Action": "sts:AssumeRoleWithSAML",
            "Condition": {
                "StringEquals": {
                    "saml:edupersonorgdn": "ExampleOrg",
                    "saml:aud": "https://region-code.signin.aws.amazon.com/saml/
acs/SAMLSP4SHN3UIS2D558H46"
                }
            }
        }
    ]
}
```

> **Note**
>
> SAML IDPs used in a role trust policy must be in the same account that the role is in.

We recommend using regional endpoints for the `saml:aud` attribute at `https://region-code.signin.aws.amazon.com/static/saml-metadata.xml`. For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints.

For the permission policy in the role, you specify permissions as you would for any role, user, or group. For example, if users from your organization are allowed to administer Amazon EC2 instances, you explicitly allow Amazon EC2 actions in the permission policy. You can do this by assigning a managed policy, such as the **Amazon EC2 Full Access** managed policy.

For details about creating a role for a SAML IdP, see Create a role for SAML 2.0 federation (console).

Finish configuration and create SAML assertions

Notify your SAML IdP that AWS is your service provider by installing the `saml-metadata.xml` file found at `https://region-code.signin.aws.amazon.com/static/saml-metadata.xml` or `https://signin.aws.amazon.com/static/saml-metadata.xml`. If SAML encryption is required, the file is found at `https://region-code.signin.aws.amazon.com/static/saml/SAMLSP4SHN3UIS2D558H46/saml-metadata.xml`.

For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints.

How you install that file depends on your IdP. Some providers give you the option to type the URL, whereupon the IdP gets and installs the file for you. Others require you to download the file from the URL and then provide it as a local file. Refer to your IdP documentation for details, or see Integrate third-party SAML solution providers with AWS for links to the web documentation for many of the supported SAML providers.

You also configure the information that you want the IdP to pass as SAML attributes to AWS as part of the authentication response. Most of this information appears in AWS as condition context keys that you can evaluate in your policies. These condition keys ensure that only authorized users in the right contexts are granted permissions to access your AWS resources. You can specify time windows that restrict when the console may be used. You can also specify the maximum time (up to 12 hours) that users can access the console before having to refresh their credentials. For details, see Configure SAML assertions for the authentication response.

View a SAML response in your browser

The following procedures describe how to view the SAML response from your service provider from in your browser when troubleshooting a SAML 2.0–related issue.

For all browsers, go to the page where you can reproduce the issue. Then follow the steps for the appropriate browser:

Topics

- Google Chrome
- Mozilla Firefox
- Apple Safari
- What to do with the Base64-encoded SAML response

Google Chrome

To view a SAML response in Chrome

These steps were tested using version 106.0.5249.103 (Official Build) (arm64) of Google Chrome. If you use another version, you might need to adapt the steps accordingly.

1. Press **F12** to start the **Developer Tools** console.
2. Select the **Network** tab, and then select **Preserve log** in the upper left of the **Developer Tools** window.
3. Reproduce the issue.
4. (Optional) If the **Method** column is not visible in the **Developer Tools Network** log pane, open the context (right-click) menu for any column label and choose **Method** to add the column.
5. Look for a **SAML Post** in the **Developer Tools Network** log pane. Select that row, and then view the **Payload** tab at the top. Look for the **SAMLResponse** element that contains the encoded request. The associated value is the Base64-encoded response.

Mozilla Firefox

To view a SAML response in Firefox

This procedure was tested on version 105.0.3 (64-bit) of Mozilla Firefox. If you use another version, you might need to adapt the steps accordingly.

1. Press **F12** to start the **Web Developer Tools** console.
2. Select the **Network** tab.

3. In the upper right of the **Web Developer Tools** window, choose options (the small gear icon). Select **Persist logs**.
4. Reproduce the issue.
5. (Optional) If the **Method** column is not visible in the **Web Developer Tools Network** log pane, open the context (right-click) menu for any column label and choose **Method** to add the column.
6. Look for a **POST SAML** in the table. Select that row, and then view the **Request** tab and find the **SAMLResponse** element. The associated value is the Base64-encoded response.

Apple Safari

To view a SAML response in Safari

These steps were tested using version 16.0 (17614.1.25.9.10, 17614) of Apple Safari. If you use another version, you might need to adapt the steps accordingly.

1. Enable Web Inspector in Safari. Open the **Preferences** window, select the **Advanced** tab, and then select **Show Develop menu in the menu bar**.
2. Now you can open Web Inspector. Choose **Develop** in the menu bar, then select **Show Web Inspector**.
3. Select the **Network** tab.
4. In the upper left of the **Web Inspector** window, choose options (the small circle icon containing three horizontal lines). Select **Preserve Log**.
5. (Optional) If the **Method** column is not visible in the **Web Inspector Network** log pane, open the context (right-click) menu for any column label and choose **Method** to add the column.
6. Reproduce the issue.
7. Look for a **POST SAML** in the table. Select that row, and then view the Headers tab.
8. Look for the **SAMLResponse** element that contains the encoded request. Scroll down to find `Request Data` with the name `SAMLResponse`. The associated value is the Base64-encoded response.

What to do with the Base64-encoded SAML response

After you find the Base64-encoded SAML response element in your browser, copy it and use your favorite Base-64 decoding tool to extract the XML tagged response.

Security tip

Because the SAML response data that you are viewing might contain sensitive security data, we recommend that you do not use an *online* base64 decoder. Instead use a tool installed on your local computer that does not send your SAML data over the network.

Built-in option for Windows systems (PowerShell):

```
PS C:
\> [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("base64encodedtext
```

Built-in option for MacOS and Linux systems:

```
$ echo "base64encodedtext" | base64 --decode
```

Review the values in the decoded file

Review the values in the decoded SAML response file.

- Verify that the value for the saml:NameID attribute matches the username for the authenticated user.
- Review the value for https://aws.amazon.com/SAML/Attributes/Role. The ARN and SAML provider are case sensitive, and the ARN must match the resource in your account.
- Review the value for https://aws.amazon.com/SAML/Attributes/RoleSessionName. The value must match the value in the claim rule.
- If you configure the attribute value for an email address or an account name, then make sure that the values are correct. The values must correspond to the email address or account name of the authenticated user.

Check for errors and confirm the configuration

Check whether the values contain errors, and confirm that the following configurations are correct.

- The claim rules meet the required elements and all ARNs are correct. For more information, see Configure your SAML 2.0 IdP with relying party trust and adding claims.

- You uploaded the latest metadata file from your IdP into AWS in your SAML provider. For more information, see Enabling SAML 2.0 federated principals to access the AWS Management Console.
- You correctly configured the IAM role's trust policy. For more information, see Methods to assume a role.

Federating AWS identities to external services

IAM outbound identity federation enables your AWS workloads to securely access external services without storing long-term credentials. Your AWS workloads can request short-lived JSON Web Tokens (JWTs) from AWS Security Token Service (AWS STS) by calling the GetWebIdentityToken API. These tokens are cryptographically signed, publicly verifiable and contain a comprehensive set of claims that assert your AWS workload's identity to external services. You can use these tokens with a wide range of third-party cloud providers, SaaS platforms, and self-hosted applications. External services verify the token's authenticity using AWS's verification keys published at well-known endpoints and use the information in the tokens to make authentication and authorization decisions.

Outbound identity federation eliminates the need to store long-term credentials such as API keys or passwords in your application code or environment variables, improving your security posture. You can control access to token generation and enforce token properties such as signing algorithms, permitted audiences and duration using IAM policies. All token requests are logged in AWS CloudTrail, providing complete audit trails for security monitoring and compliance reporting. You can also customize tokens with tags that appear as custom claims, enabling external services to implement fine-grained, attribute-based access control.

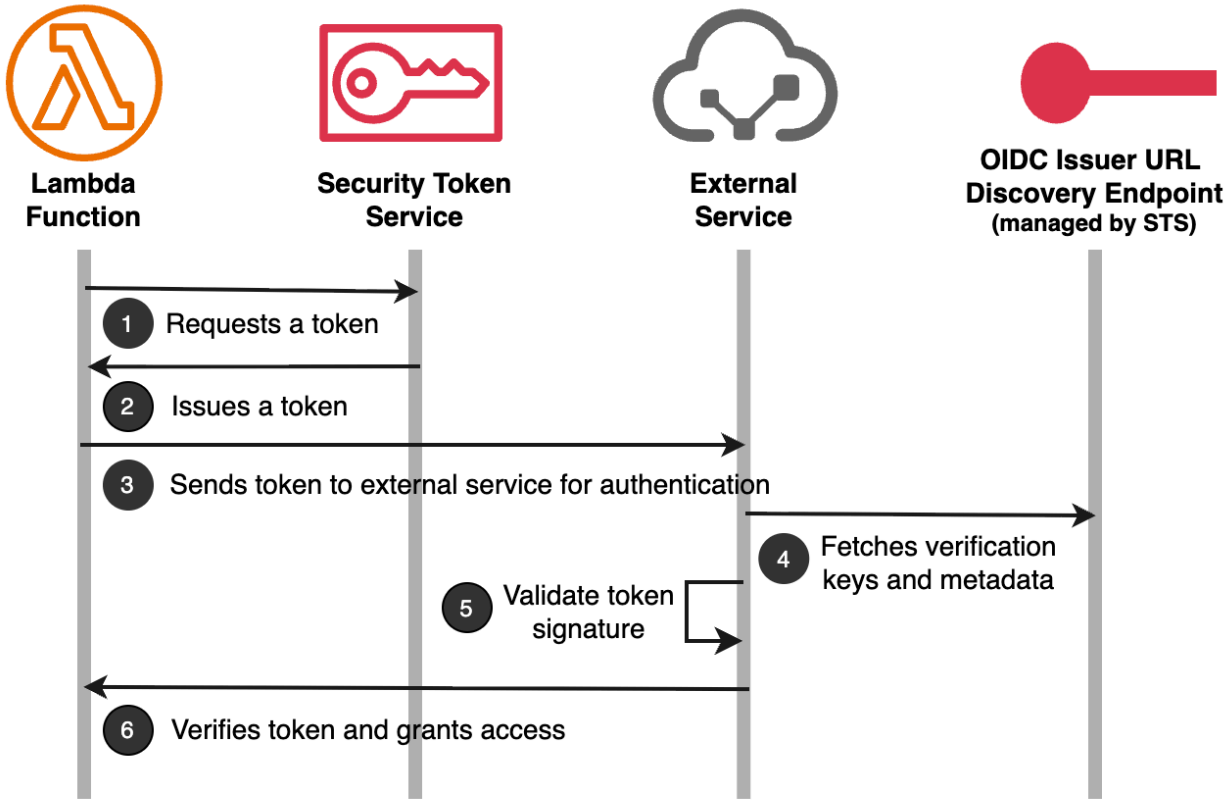
For instructions on setting up outbound federation, see Getting started with outbound identity federation. For more information about token contents, see Understanding token claims. For instructions on controlling which workloads can request tokens, see Controlling access with IAM policies. To use these tokens to assume a role in a different AWS partition, see Federating AWS identities across AWS partitions.

Common use cases

Using outbound identity federation, your AWS workloads can securely:

- Access resources and services in external cloud providers. For example, a Lambda function processing data can write results to an external cloud provider's storage service or query their database.
- Integrate with external software-as-a-service (SaaS) providers for analytics, data processing, monitoring etc. For example, your Lambda functions can send metrics to observability platforms.
- Authenticate with your own applications hosted on AWS, external cloud providers or on-premises data centers, enabling secure hybrid and multi-cloud architectures. For example, your AWS workloads can interact with containerized applications running in your on-premises Kubernetes cluster.

How It Works



1. The Lambda function calls the GetWebIdentityToken API to request a JSON Web Token (JWT) from AWS Security Token Service (AWS STS).
2. AWS STS validates the request and returns a signed JWT to the Lambda function.
3. The Lambda function sends the JWT to the external service.

4. The external service extracts the issuer URL from the token, verifies it matches a known trusted issuer, and fetches AWS's verification keys and metadata from the OIDC discovery endpoint.
5. The external service uses the verification keys to verify the token's signature and validates claims such as expiration time, subject and audience.
6. After successful validation, the external service grants access to the Lambda function.

Getting started with outbound identity federation

This guide shows you how to enable outbound identity federation for your AWS account and obtain your first JSON Web Token (JWT) (using the GetWebIdentityToken API). You will enable the feature, establish a trust relationship with an external service, configure IAM permissions, and request a token using either AWS CLI or AWS SDK for Python (Boto3).

Prerequisites

Before you begin, make sure you have:

- Latest version of the AWS CLI or Python 3.8 (or later) and Boto3 installed (for AWS SDK examples)
- An external service account where you can configure trust relationships (such as an external cloud provider, SaaS provider, or a test application)

Note

- The `GetWebIdentityToken` API is not available on the STS Global endpoint.
- To assume a role in a different AWS partition, exchange the JWT returned by GetWebIdentityToken for temporary credentials by calling AssumeRoleWithWebIdentity. This exchange is supported only across partition boundaries. For more information, see Federating AWS identities across AWS partitions. For same-partition role assumption, use the AssumeRole API instead.

Enable outbound identity federation for your account

You must enable outbound identity federation before you can request tokens. You can enable the feature using the AWS Management Console or programmatically using the EnableOutboundWebIdentityFederation API.

Using the AWS CLI

```
aws iam enable-outbound-web-identity-federation
```

Using AWS SDK for Python

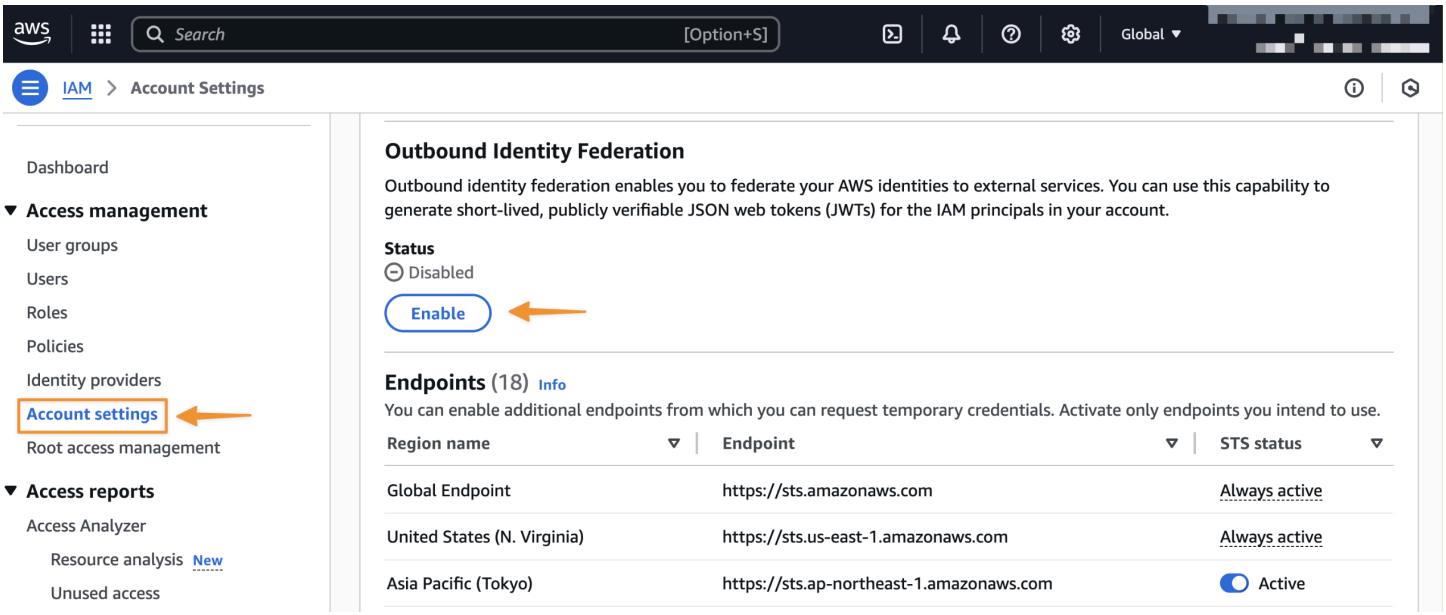
```
import boto3

# Create IAM client
iam_client = boto3.client('iam')

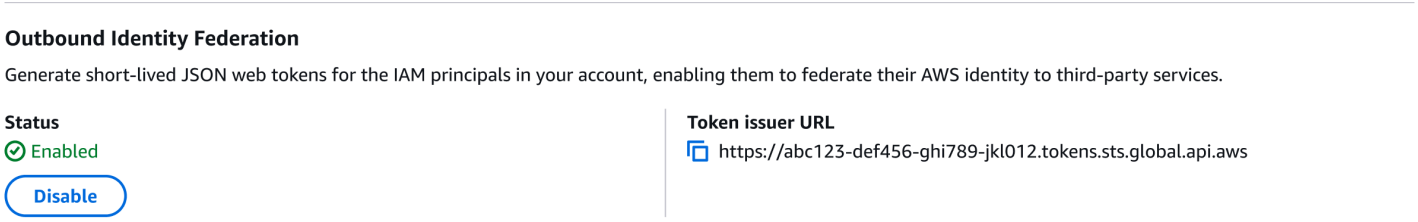
# Enable outbound identity federation
response = iam_client.enable_outbound_web_identity_federation()
print(f"Feature enabled. Issuer URL: {response['IssuerUrl']}")
print(f"Status: {response['Status']}")
```

Using the AWS Console

Navigate to IAM and select **Account Settings** under the **Access Management** section of the left-hand navigation menu



Once you enable the feature, make note of your account-specific issuer URL. You will use this URL when configuring trust relationships in external services. You can also retrieve this issuer URL as needed using the GetOutboundWebIdentityFederationInfo API.



Establish trust relationship in external service

Configure the external service to trust and accept tokens issued by your AWS account. The specific steps vary by service, but generally involve:

- Registering your AWS account issuer URL as a trusted identity provider
- Configuring which claims to validate (audience, subject patterns)
- Mapping token claims to permissions in the external service

Refer to the external service documentation for detailed configuration instructions. If the relying party is AWS in another partition, see Federating AWS identities across AWS partitions.

Configure IAM permissions

Create an IAM policy that grants permission to call the GetWebIdentityToken API and attach the policy to an IAM role that needs to generate tokens.

This example policy grants access to token generation with specific restrictions. It allows requesting tokens only for "https://api.example.com" as the audience and enforces a maximum token lifetime of 5 minutes (300 seconds). Refer to the section called “IAM condition keys” for a list of condition keys you can use to enforce token properties.

Example IAM policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "sts:GetWebIdentityToken",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "sts:IdentityTokenAudience": "https://api.example.com"
                },
                "Null": {
                    "sts:IdentityTokenAudience": "false"
                },
                "NumericLessThanEquals": {
                    "sts:DurationSeconds": 300
                }
            }
        }
    ]
}
```

Request your first JSON Web Token (JWT)

You can request a JSON Web Token using the GetWebIdentityToken API. You can specify the following parameters when calling the API:

- **Audience (required):** The intended recipient of the token. This value populates the "aud" claim in the JWT. External services validate this claim to make sure the token was intended for them.
- **SigningAlgorithm (required):** The cryptographic algorithm used to sign the token. Valid values are ES384 and RS256. Use ES384 for optimal security and performance, or RS256 for broader compatibility with systems that do not support ECDSA.
- **DurationSeconds (optional):** The token lifetime in seconds. Valid values range from 60 to 3600. The default is 300 (5 minutes). We recommend shorter token lifetimes for increased security.
- **Tags (optional):** A list of key-value pairs to include as custom claims in the token. External services can use these claims for fine-grained authorization.

The API returns the following fields:

- **WebIdentityToken:** The signed JWT as a base64url-encoded string. Include this token in requests to external services.
- **Expiration:** The UTC timestamp for when the token expires.

Using the AWS CLI

```
aws sts get-web-identity-token \
    --region us-east-1 \
    --endpoint-url https://sts.us-east-1.amazonaws.com \
    --audience "https://api.example.com" \
    --signing-algorithm ES384 \
    --duration-seconds 300 \
    --tags Key=team,Value=data-engineering \
           Key=environment,Value=production \
           Key=cost-center,Value=analytics
```

Using AWS SDK for Python

```
import boto3

# GetWebIdentityToken requires a Regional STS endpoint.
sts_client = boto3.client(
    'sts',
    region_name='us-east-1',
    endpoint_url='https://sts.us-east-1.amazonaws.com'
)

response = sts_client.get_web_identity_token(
    Audience=['https://api.example.com'],
    DurationSeconds=300,
    SigningAlgorithm='ES384',
    Tags=[
        {'Key': 'team', 'Value': 'data-engineering'},
        {'Key': 'environment', 'Value': 'production'},
        {'Key': 'cost-center', 'Value': 'analytics'}
    ]
)

token = response['WebIdentityToken']
```

You can also decode the JWT to inspect its contents using standard JWT libraries like PyJWT, Python-jose for Python, Nimbus JOSE+JWT for Java or debuggers like jwt.io. Refer to the section called “Understanding token claims” for more information on claims included in the token.

Use the token with an external service

After receiving the token, include it in requests to the external service. The method varies by service, but most services accept tokens in the Authorization header. The external service should implement token validation logic that fetches the JWKS keys from your issuer's well-known endpoint, verifies the token's signature, and validates essential claims before granting access to your AWS workloads.

Fetch verification keys and metadata from the OpenID Connect (OIDC) endpoints

The unique issuer URL for your AWS account hosts the OpenID Connect (OIDC) discovery endpoints that contain verification keys and metadata necessary for token verification.

The OIDC discovery endpoint URL contains metadata some providers use to verify tokens. It is available at:

```
{issuer_url}/.well-known/openid-configuration
```

The JWKS (JSON Web Key Set) endpoint contains keys used to verify token signatures. It is available at:

```
{issuer_url}/.well-known/jwks.json
```

Fetch JWKS using curl

```
curl https://{issuer_url}/.well-known/jwks.json
```

Response:

```
{
  "keys": [
    {
      "kty": "EC",
      "use": "sig",
      "kid": "key-id-1",
      "alg": "ES384",
      "crv": "P-384",
      "x": "base64-encoded-x-coordinate",
      "y": "base64-encoded-y-coordinate"
```

```
    },
    {
      "kty": "RSA",
      "use": "sig",
      "kid": "key-id-2",
      "n": "base64-encoded-modulus",
      "e": "AQAB"
    }
  ]
}
```

Using AWS SDK for Python

```
import requests

# Fetch Openid Configuration
open_id_config_response = requests.get("https://{issuer_url}/.well-known/openid-
configuration")
open_id_config = open_id_config_response.json()

# Fetch JWKS
jwks_response = requests.get("https://{issuer_url}/.well-known/jwks.json")
jwks = jwks_response.json()
```

We recommend caching these keys to avoid fetching them for every token verification.

Essential claim validations

- **Subject (sub):** Verify the subject claim contains the expected IAM principal ARN pattern.
- **Expiration (exp):** Make sure the token has not expired. JWT libraries typically handle this automatically.
- **Audience (aud):** Verify the audience matches your expected value. This prevents tokens intended for other services from being used with yours.
- **Issuer (iss):** Verify the issuer matches the AWS account(s) you trust. Maintain a list of trusted issuer URLs.

Whenever possible, you should validate additional AWS-specific claims to implement fine-grained access control in your external service. For example, validate the org_id claim to restrict access to IAM principals in your AWS Organization, check principal_tags to enforce attribute-based access control (such as allowing only production environments or specific teams), or verify session context

claims like lambda_source_function_arn or ec2_instance_source_vpc to restrict access based on the compute resource. Refer to Understanding token claims for a full list of claims included in the token.

Understanding token claims

When you call the GetWebIdentityToken API, AWS Security Token Service returns a signed JSON Web Token (JWT) that contains a set of claims that represent the identity of the IAM principal. These tokens are compliant with RFC 7519. Understanding the structure and contents of these tokens helps you implement secure authentication flows, configure appropriate claim validations in external services, and effectively use custom claims for fine-grained access control.

The JWT includes standard OpenID Connect (OIDC) claims such as subject ("sub"), audience ("aud"), issuer ("iss") to facilitate interoperability across different external services. AWS STS populates the token with AWS identity-specific claims (like the AWS Account ID and Principal tags) and session context claims (like EC2 instance ARNs) when applicable. You can also add custom claims to the token by passing them as request tags to the GetWebIdentityToken API. The AWS identity-specific claims, and session context claims and custom claims are nested under the "https://sts.amazonaws.com/" namespace in the token.

Refer to the sample token below for a list of claims included in the token. Please note that all these claims may not be present in a token at the same time.

```
{
  "iss": "https://abc123-def456-ghi789-jkl012.tokens.sts.global.api.aws",
  "aud": "https://api.example.com",
  "sub": "arn:aws:iam::123456789012:role/DataProcessingRole",
  "iat": 1700000000,
  "exp": 1700000900,
  "jti": "xyz123-def456-ghi789-jkl012",
  "https://sts.amazonaws.com/": {
    "aws_account": "123456789012",
    "source_region": "us-east-1",
    "principal_id": "arn:aws:iam::123456789012:role/DataProcessingRole",
    "org_id": "o-abc1234567",
    "ou_path": [
      "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/"
    ],
    "principal_tags": {
      "environment": "production",
      "team": "data-engineering",
```

```
      "cost-center": "engineering"
    },
    "lambda_source_function_arn": "arn:aws:lambda:us-
east-1:123456789012:function:process-data",
    "request_tags": {
        "job-id": "job-2024-001",
        "priority": "high",
        "data-classification": "sensitive"
    }
  }
}
```

Standard claims

The standard OIDC claims present in the tokens facilitate interoperability with a wide range of external services. These claims can be validated using most JWT libraries.

Claim	Name	Description	Example Value
iss	Issuer	Your account-specific issuer URL. External services validate this claim to ensure it matches their trusted issuer.	https://abc123-def456-ghi789-jkl012.tokens.sts.global.api.aws
aud	Audience	The intended recipient for the token specified in the GetWebIdentityToken request.	https://api.example.com
sub	Subject	The ARN of the IAM principal that requested the token.	arn:aws:iam::123456789012:role/DataProcessingRole
iat	Issued At	NumericDate value that identifies the time at which the JWT was issued.	1700000000
exp	Expiration	NumericDate value that identifies the expiration time on or after which the JWT MUST NOT be accepted for processing.	1700000900

Claim	Name	Description	Example Value
jti	JWT ID	Unique identifier for this token instance.	xyz123-def456-ghi789-jkl012

Custom claims

In addition to the standard OIDC claims, AWS STS adds claims about the identity and session context when applicable. You can also add your own claims to the token by passing them as request tags. Custom claims are nested under the https://sts.amazonaws.com/ namespace.

The condition keys in the following tables identify the source request context used to populate the claims. For the condition keys used in cross-partition role assumption and their availability in the destination session, see the AWS STS tab under Available keys for AWS OIDC federation.

AWS identity claims

These claims provide detailed information about your AWS account, organization structure, and IAM principal.

Claim	Description	Source condition key	Example Value
aws_account	Your AWS account ID	aws:PrincipalAccount	123456789012
source_region	The AWS region where the token was requested	aws:RequestedRegion	us-east-1
principal_id	The ARN of the IAM principal that requested the token.	aws:PrincipalArn	arn:aws:iam::123456789012:role/DataProcessingRole
org_id	Your AWS Organizations ID (if your account is part of an organization)	aws:PrincipalOrgID	o-abc1234567

Claim	Description	Source condition key	Example Value
ou_path	The AWS Organizations paths for the source account, as an array. Present when organization paths are available in the source request context.	aws:PrincipalOrgPaths	["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/"]
principal_tags	Tags attached to the IAM principal or assumed role session. When a token is requested where the requesting IAM principal has both principal tags and session tags, the session tags will be present in the JWT.	aws:PrincipalTag/<tag-key>	{"environment": "production", "team": "data-engineering", "cost-center":"engineering"}

Session context claims

These claims provide information about the compute environment and session where the token request originated. AWS STS automatically includes these claims when applicable based on the requesting principal's session context.

Claim	Description	Source condition key	Example Value
original_session_exp	When the original role session credentials will expire (for assumed roles)	N/A	2024-01-15T10:00:00Z
invoked_by	The AWS service acting on the customer's behalf by assuming the source role, or the last service in a forward access session. This claim is present only when the source	N/A	ec2.amazonaws.com

Claim	Description	Source condition key	Example Value
	session includes that service's identity.		
mfa_present	Indicates that the IAM principal authenticated with MFA. Present only when MFA was used, with the string value `"true"`. It is never `"false"`.	aws:MultiFactorAuthPresent	"true"
mfa_age	The number of seconds since the IAM principal authenticated with MFA, encoded as a string. Present only when MFA was used and the authentication age is available.	aws:MultiFactorAuthAge	"1800"
federated_provider	The identity provider name for federated sessions	aws:FederatedProvider	arn:aws:iam::111122223333:oidc-provider/your_oidc_provider
identity_store_user_id	IAM Identity Center user ID	identitystore:UserId	user-abc123def456
identity_store_arn	ARN of the Identity Center identity store	identitystore:IdentityStoreArn	arn:aws:identitystore::123456789012:identitystore/d-abc1234567

Claim	Description	Source condition key	Example Value
ec2_source_instance_arn	ARN of the requesting EC2 instance	ec2:SourceInstanceArn	arn:aws:ec2:us-east-1:123456789012:instance/i-abc123def456
ec2_instance_source_vpc	VPC ID where EC2 role credentials were delivered	aws:Ec2InstanceSourceVpc	vpc-abc123def456
ec2_instance_source_private_ipv4	Private IPv4 address of the EC2 instance	aws:Ec2InstanceSourcePrivateIPv4	10.0.1.25
ec2_role_delivery	The instance metadata service (IMDS) version that delivered the source role credentials: `1.0` for IMDSv1 or `2.0` for IMDSv2.	ec2:RoleDelivery	2.0
source_identity	Source identity set by the principal	aws:SourceIdentity	admin-user
lambda_source_function_arn	ARN of the calling Lambda function	lambda:SourceFunctionArn	arn:aws:lambda:us-east-1:123456789012:function:my-function
glue_credential_issuing_service	AWS Glue service identifier for Glue jobs	glue:CredentialIssuingService	glue.amazonaws.com

Request tags

You can add custom claims to tokens by specifying tags in the GetWebIdentityToken API request. These claims appear under the request_tags field in the token and enable you to pass specific

information that external services can use for fine-grained authorization decisions. You can specify up to 50 tags per request.

Example request:

```
response = sts_client.get_web_identity_token(
    Audience=['https://api.example.com'],
    SigningAlgorithm='ES384',
    Tags=[
        {'Key': 'team', 'Value': 'data-engineering'},
        {'Key': 'cost-center', 'Value': 'analytics'},
        {'Key': 'environment', 'Value': 'production'}
    ]
)
```

Resulting claims in token:

```
{
  "request_tags": {
    "team": "data-engineering",
    "cost-center": "analytics",
    "environment": "production"
  }
}
```

Controlling access with IAM policies

IAM provides multiple policy types to control access to the outbound identity federation feature. You can use identity-based policies to control which IAM principals can request tokens and enforce specific token properties such as audience, lifetimes, and signing algorithms. Service Control Policies (SCPs) enable you to enforce organization-wide restrictions on token generation across all accounts in your AWS Organizations. Resource Control Policies (RCPs) control access at the resource level. You can also use VPC endpoint policies to restrict which principals can access the AWS STS `GetWebIdentityToken` API through your VPC endpoints, adding network-level controls to your security posture. This section explains how to implement fine-grained access controls using these policy types and condition keys.

To request identity tokens, an IAM principal must have the `sts:GetWebIdentityToken` permission. Grant this permission through identity policies attached to IAM users or roles. To allow

Tags (key, value pairs) to be passed to the GetWebIdentityToken call, the IAM principal must have the `sts:TagGetWebIdentityToken` permission.

- Use the sts:IdentityTokenAudience condition key to restrict the audiences for which a principal can request tokens.
- Use the sts:DurationSeconds condition key to enforce maximum token lifetimes.
- Use the sts:SigningAlgorithm condition key to require specific cryptographic algorithms.
- Use the aws:RequestTag condition key compare the tag key-value pair that was passed in the request with the tag pair that you specify in the policy.
- Use the aws:TagKeys condition key to compare the tag keys in a request with the keys that you specify in the policy.

Refer to IAM and AWS STS condition keys to learn more about the condition keys available in IAM policies.

Use `ForAllValues:StringEquals` to require every requested audience to match an allowed value. Include a `Null` condition with `false` to require `sts:IdentityTokenAudience` to be present and nonempty. For more information, see Set operators for multivalued context keys.

The following identity-based policy allows tokens only for the listed audiences, with a lifetime of up to 300 seconds and ES384 signing.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowTokenGenerationWithRestrictions",
            "Effect": "Allow",
            "Action": "sts:GetWebIdentityToken",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "sts:IdentityTokenAudience": [
                        "https://api1.example.com",
                        "https://api2.example.com"
                    ]
                },
                "Null": {
                    "sts:IdentityTokenAudience": "false"
```

```
                },
                "NumericLessThanEquals": {
                    "sts:DurationSeconds": 300
                },
                "StringEquals": {
                    "sts:SigningAlgorithm": "ES384"
                }
            }
        }
    ]
}
```

Best practices

Follow these recommendations to securely federate your AWS identities to external services.

- **Use short token lifetimes:** Request tokens with the shortest lifetime that meets your operational needs.
- **Implement least privilege access and restrict token properties with IAM policies:** Grant the `sts:GetWebIdentityToken` permission only to IAM principals that require it. Use condition keys to specify signing algorithms, permitted token audiences, and maximum token lifetimes as you require.
- **Validate claims in external services:** For security, always validate relevant claims such as subject ("sub"), audience ("aud") etc. to ensure they match your expected values. Validate custom claims when possible to enable fine-grained authorization decisions in external services.

Federating AWS identities across AWS partitions

Cross-partition identity federation allows a workload to obtain short-lived access across AWS partitions. AWS partitions are hard isolation boundaries with independent instances of AWS Identity and Access Management (IAM), so a credential issued in one partition isn't valid in another. As a result, direct role assumption with the AssumeRole API doesn't work across partition boundaries.

Cross-partition identity federation works as a two-step exchange: your workload calls sts:GetWebIdentityToken to obtain a short-lived JSON Web Token (JWT) from AWS Security Token Service (STS) in the source partition. It then calls sts:AssumeRoleWithWebIdentity to exchange the token for temporary credentials in the destination partition. This approach lets the workload

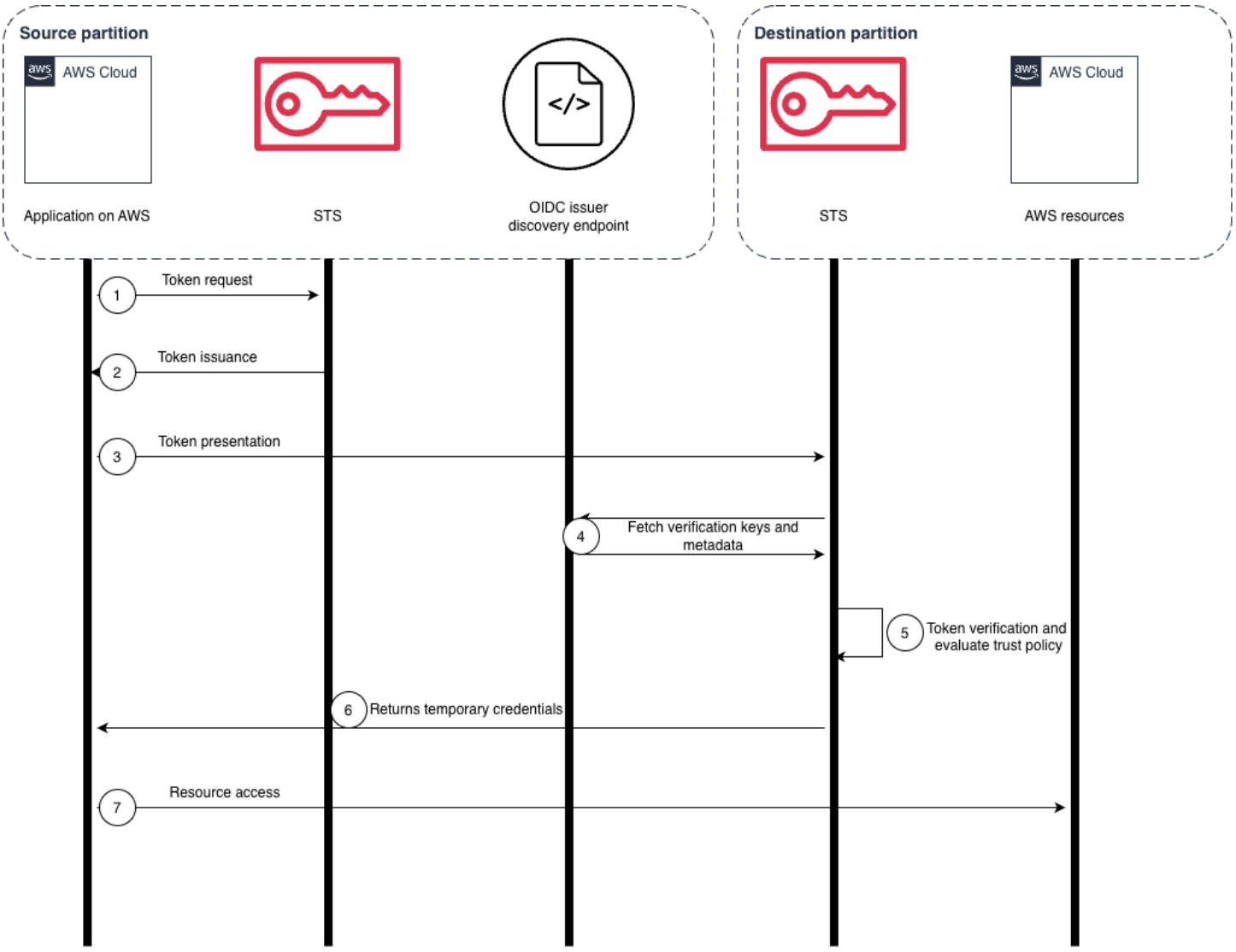
use temporary credentials in both partitions, following the IAM best practice of using temporary credentials for workloads.

Note

- Cross-partition identity federation is available in the commercial AWS Regions (`aws`), the China Regions (`aws-cn`), the AWS GovCloud (US) Regions (`aws-us-gov`), and the AWS European Sovereign Cloud (`aws-eusc`).
- The destination role must be in a different AWS partition from the one where the token was issued. The JWTs from `GetWebIdentityToken` cannot be used for OpenID Connect (OIDC) federation into the same partition through `AssumeRoleWithWebIdentity`. For role assumption within a partition, use the `AssumeRole` API instead.
- If your workload requires cross-partition access, consult your compliance team to make sure that your implementation aligns with your regulatory or compliance obligations.

How cross-partition identity federation works

Your workload first requests a token from AWS STS in the source partition, then sends the token to AWS STS in the destination partition. To validate the request, AWS STS in the destination partition retrieves verification keys from the source account's OIDC issuer, verifies the token, and evaluates the destination role's trust policy. If the request is allowed, AWS STS returns temporary credentials to the workload, which uses them to call services and access resources in the destination partition.



The numbered steps that follow describe each request and response in the diagram.

The flow works as follows:

1. **Token request**: Your workload, running in an AWS Region in the source partition (for example, `us-east-1` in the `aws` partition), calls the `sts:GetWebIdentityToken` API.
2. **Token issuance**: AWS STS returns a short-lived, cryptographically signed JWT. The token contains claims that assert your workload's AWS identity, including the account ID, the principal Amazon Resource Name (ARN), the source AWS Region, and more.
3. **Token presentation**: Your workload calls `sts:AssumeRoleWithWebIdentity` on the AWS STS endpoint in an AWS Region in the destination partition (for example, `cn-north-1` in the `aws-cn` partition), passing the JWT as the web identity token and the ARN of the target role in the destination partition.

4. **Fetch verification keys and metadata**: AWS STS in the destination partition retrieves the public verification key from the JWKS endpoint of the source account's issuer URL.
5. **Validation**: AWS STS in the destination partition verifies the token's signature and evaluates the target role's trust policy.
6. **Return temporary credentials**: AWS STS in the destination partition returns temporary credentials scoped by the policies attached to the role.
7. **Cross-partition resource access**: Your workload uses the temporary credentials to access resources in the destination partition, such as an Amazon S3 bucket.

For more information, see the following topics:

- Setting up cross-partition identity federation
- Controls for cross-partition identity federation
- Auditing cross-partition identity federation
- Available keys for AWS OIDC federation

Setting up cross-partition identity federation

Cross-partition identity federation works as a two-step exchange: your workload calls `sts:GetWebIdentityToken` to obtain a JWT from AWS STS in the source partition, then exchanges that token for temporary credentials from AWS STS in the destination partition by calling `sts:AssumeRoleWithWebIdentity`.

Establishing this flow involves a one-time setup phase and a runtime phase that repeats each time the workload needs credentials in the destination partition. This page explains how to configure the source and destination accounts.

Setup phase

1. In the source partition, enable outbound identity federation for your account. For instructions, see the section called “Enable outbound identity federation for your account”.
2. In the source partition, create an IAM principal or choose an existing one, and grant it permission to call `sts:GetWebIdentityToken`. Use IAM policy conditions to restrict the permitted audiences, token duration, and signing algorithms. For instructions, see Controlling access with IAM policies.

3. In the destination partition, create an OpenID Connect (OIDC) identity provider using the source account's issuer URL. Make sure the provider's client ID list includes the audience your workload will use with `GetWebIdentityToken`. For instructions, see Create an OpenID Connect (OIDC) identity provider in IAM.
4. In the destination partition, create an IAM role or choose an existing one for the source IAM principal to assume. Configure the role's trust policy to trust the OIDC identity provider you created and allow `sts:AssumeRoleWithWebIdentity`. For instructions, see Who can assume a role.

Runtime phase

1. Using credentials for the IAM principal in the source partition, call sts:GetWebIdentityToken with the audience you configured in the destination account's OIDC identity provider.

   > **Note**
   >
   > For cross-partition identity federation, specify exactly one audience when calling `GetWebIdentityToken`. Tokens requested with multiple audiences can't be exchanged for temporary credentials using `AssumeRoleWithWebIdentity`.

2. AWS STS in the source partition returns a short-lived JWT that represents the source IAM principal's identity.
3. Call sts:AssumeRoleWithWebIdentity on the AWS STS endpoint in the destination partition, passing the JWT, the ARN of the destination role you configured during setup, and a role session name (`RoleSessionName`).
4. AWS STS in the destination partition validates the JWT and evaluates the destination role's trust policy. If the request is allowed, AWS STS returns temporary credentials for that role.
5. Use the temporary credentials to access AWS resources or communicate with an AWS service in the destination partition.

The following AWS CLI example shows the runtime phase, using `us-east-1` in the `aws` partition as the source Region and `cn-north-1` in the `aws-cn` partition as the destination Region. Use credentials for the source IAM principal to call `GetWebIdentityToken`. The `AssumeRoleWithWebIdentity` call authenticates using the JWT returned by `GetWebIdentityToken` and doesn't require AWS credentials.

```
# Source partition (aws): get a short-lived JWT
TOKEN=$(aws sts get-web-identity-token \
  --region us-east-1 \
  --endpoint-url https://sts.us-east-1.amazonaws.com \
  --audience "your-audience" \
  --signing-algorithm ES384 \
  --duration-seconds 300 \
  --query WebIdentityToken --output text)

# Destination partition (aws-cn): exchange it for temporary credentials
aws sts assume-role-with-web-identity \
  --region cn-north-1 \
  --role-arn arn:aws-cn:iam::111122223333:role/DestinationWorkloadRole \
  --role-session-name cross-partition-session \
  --web-identity-token "$TOKEN" \
  --no-sign-request
```

The following example shows the same flow using the AWS SDK for Python (Boto3).

```
import boto3
from botocore import UNSIGNED
from botocore.config import Config

# Source partition (aws): get a short-lived JWT from the Regional STS endpoint
source_sts = boto3.client(
    'sts',
    region_name='us-east-1',
    endpoint_url='https://sts.us-east-1.amazonaws.com'
)
token = source_sts.get_web_identity_token(
    Audience=['your-audience'],
    SigningAlgorithm='ES384',
    DurationSeconds=300
)['WebIdentityToken']

# Destination partition (aws-cn): exchange it for temporary credentials.
# AssumeRoleWithWebIdentity doesn't require AWS credentials.
destination_sts = boto3.client(
    'sts',
    region_name='cn-north-1',
    config=Config(signature_version=UNSIGNED)
)
credentials = destination_sts.assume_role_with_web_identity(
```

```
    RoleArn='arn:aws-cn:iam::111122223333:role/DestinationWorkloadRole',
    RoleSessionName='cross-partition-session',
    WebIdentityToken=token
)['Credentials']
```

Controls for cross-partition identity federation

Cross-partition identity federation involves three access decisions: who can request a token in the source account, which JWTs can be used to assume a role in the destination account, and what the resulting role session can access. IAM and AWS Organizations policies govern these decisions. This page explains where each policy applies and provides examples for controlling access at each stage.

Where to apply access controls

- Control token requests in the source partition
- Control access in the destination partition
  - Who can assume a role
  - What role sessions can do

Control token requests in the source partition

In the source account, IAM identity-based policies grant permission to call `sts:GetWebIdentityToken` and can restrict token audiences, lifetimes, and signing algorithms. Token requests that include tags also require the `sts:TagGetWebIdentityToken` permission. Permissions boundaries, session policies, and applicable service control policies (SCPs) can further restrict permission to request tokens. Requests made through an AWS STS VPC endpoint must also be allowed by its endpoint policy.

For condition keys and policy examples that control token requests, see Controlling access with IAM policies.

> **Note**
>
> Request the token with only the audience configured for your destination. `AssumeRoleWithWebIdentity` rejects tokens from `GetWebIdentityToken` that have more than one audience. See Setting up cross-partition identity federation.

Control access in the destination partition

In the destination account, the role's trust policy controls which JWTs can be used to assume the role through `sts:AssumeRoleWithWebIdentity`. Applicable resource control policies (RCPs) can restrict role assumption across destination member accounts.

After role assumption, identity-based policies and resource-based policies grant the session access to AWS services and resources. Permissions boundaries, session policies, and applicable service control policies (SCPs) and RCPs can further restrict what the role session can access. Requests made through a service's VPC endpoint must also be allowed by its endpoint policy.

Who can assume a role

The following examples use the destination role's trust policy to allow a source role to assume it, and an RCP to allowlist trusted issuers or source partitions. For supported token claim condition keys, see the AWS STS tab under Available keys for AWS OIDC federation.

Example: Allow a source role to assume the destination role

The destination role's trust policy allows the specified source role to assume it when the token has the expected audience and was issued in `us-east-1`. The source partition is `aws` and the destination partition is `aws-cn`. Replace the issuer host in the provider ARN and condition keys with your source account's issuer host, and replace `your-audience` with the audience configured for your destination OIDC provider. Replace `us-east-1` with the source AWS Region you want to allow.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws-cn:iam::111122223333:oidc-
provider/a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:sub":
 "arn:aws:iam::123456789012:role/SourceWorkloadRole",
          "a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:aud": "your-
audience",
```

```
 "a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:source_region": "us-
east-1"
        }
      }
    }
  ]
}
```

Note

The sub condition matches every session of SourceWorkloadRole. To limit access to a specific Amazon EC2 instance or Lambda function, use *issuer-host*:ec2_source_instance_arn or *issuer-host*:lambda_source_function_arn, respectively, when those claims are present. Use *issuer-host*:org_id to require a specific source organization.

Example: Allowlist trusted issuers with a resource control policy (RCP)

The aws:FederatedProvider condition key identifies the OIDC provider in the destination account. The issuer host in that provider's ARN identifies the source account.

The following RCP enforces an allowlist of approved source issuers for cross-partition role assumption. The StringLike condition scopes the deny to OIDC provider ARNs that match the AWS STS issuer pattern. Replace the example issuer host with your approved source issuer host.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceIssuerAllowlist",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:FederatedProvider": "arn:*:iam::*:oidc-provider/*.tokens.sts.*"
        },
        "StringNotLike": {
```

```
          "aws:FederatedProvider": [
            "arn:*:iam::*:oidc-
provider/a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws"
          ]
        }
      }
    }
  ]
}
```

Example: Allowlist trusted source partitions with a resource control policy (RCP)

To include all issuers from an approved source partition, match the issuer host suffix instead of an account-specific issuer host.

Source partition	Issuer host suffix
aws	tokens.sts.global.api.aws
aws-us-gov	tokens.sts.us-gov.api.aws
aws-cn	tokens.sts.global.api.amazonwebservices.com.cn
aws-eusc	tokens.sts.global.api.amazonwebservices.eu

The following RCP enforces an allowlist of approved source partitions for cross-partition role assumption. This example includes the `aws` partition in the allowlist. The `StringLike` condition scopes the deny to OIDC provider ARNs that match the AWS STS issuer pattern. The wildcard before the issuer host suffix matches every account's issuer in the approved partition. Add an ARN pattern for each approved source partition.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceSourcePartitionAllowlist",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sts:AssumeRoleWithWebIdentity",
```

```
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:FederatedProvider": "arn:*:iam::*:oidc-provider/*.tokens.sts.*"
        },
        "StringNotLike": {
          "aws:FederatedProvider": [
            "arn:*:iam::*:oidc-provider/*.tokens.sts.global.api.aws"
          ]
        }
      }
    }
  ]
}
```

What role sessions can do

The following examples use an SCP to block requests by source issuer and an RCP to block Amazon S3 access by source AWS Region. After a policy change propagates, these denies apply to matching requests from new and existing role sessions, including sessions whose temporary credentials haven't expired. For the condition keys used in these examples, see aws:FederatedProvider and aws:CrossPartitionSourceRegion.

Example: Block requests by source issuer with an SCP

This SCP denies requests from destination role sessions that federated through the specified source issuer. It applies to destination roles in member accounts covered by the SCP. The example uses destination accounts in the `aws-cn` partition. Replace the example issuer host with the source issuer host you want to block.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenySessionsFromSourceIssuer",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:FederatedProvider": "arn:aws-cn:iam::*:oidc-
provider/a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws"
```

```
        }
      }
    }
  ]
}
```

Example: Block Amazon S3 access by source Region with an RCP

This RCP denies access to Amazon S3 resources in your AWS organization's member accounts covered by the policy when a request uses a cross-partition federated session whose source token was issued in `us-east-1`. Replace `us-east-1` with the source AWS Region you want to block. Both SCPs and RCPs support the `aws:FederatedProvider` and `aws:CrossPartitionSourceRegion` condition keys.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyS3AccessFromSourceRegion",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CrossPartitionSourceRegion": "us-east-1"
        }
      }
    }
  ]
}
```

Auditing cross-partition identity federation

The `GetWebIdentityToken` CloudTrail event captures the token request in the source account. The `AssumeRoleWithWebIdentity` CloudTrail event captures the role assumption request in the destination account. Match `webIdentityTokenId` values to connect the source and destination events, and use `sourcePrincipalArn` and `sourceRegion` to identify who requested the token and where it was issued.

For cross-partition identity federation, the events include the following information:

- `webIdentityTokenId` appears under `responseElements` in the source event and under `requestParameters` in the destination event.
- `additionalEventData.crossPartitionData.sourcePrincipalArn` in the destination event identifies the ARN of the source IAM principal that requested the token.
- `additionalEventData.crossPartitionData.sourceRegion` in the destination event identifies the source AWS Region where AWS STS issued the token.

Use these fields together to trace a destination role assumption back to the source token request. Matching `webIdentityTokenId` values connect the `AssumeRoleWithWebIdentity` event to the corresponding `GetWebIdentityToken` event. The `sourcePrincipalArn` and `sourceRegion` fields identify who requested the token and where it was issued.

Example: Token request in the source account

The following example shows a `GetWebIdentityToken` event in the source account.


```
{
  "eventVersion": "1.11",
  "userIdentity": {
      "type": "AssumedRole",
      "principalId": "AROA123456789EXAMPLE:workload-session",
      "arn": "arn:aws:sts::123456789012:assumed-role/SourceWorkloadRole/workload-
session",
      "accountId": "123456789012",
      "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
      "sessionContext": {
          "sessionIssuer": {
              "type": "Role",
              "principalId": "AROA123456789EXAMPLE",
              "arn": "arn:aws:iam::123456789012:role/SourceWorkloadRole",
              "accountId": "123456789012",
              "userName": "SourceWorkloadRole"
          },
          "attributes": {
              "creationDate": "2026-10-02T01:03:26Z",
              "mfaAuthenticated": "false"
          }
      }
  },
  "eventTime": "2026-10-02T01:05:02Z",
  "eventSource": "sts.amazonaws.com",
```

```
  "eventName": "GetWebIdentityToken",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "aws-cli/2.34.4 md/command#sts.get-web-identity-token",
  "requestParameters": {
      "audience": [
          "your-audience"
      ],
      "durationSeconds": 300,
      "signingAlgorithm": "ES384"
  },
  "responseElements": {
      "webIdentityTokenId": "6e70a1d6-7cbd-44f5-9b94-d0540EXAMPLE",
      "expiration": "2026-10-02T01:10:02Z"
  },
  "additionalEventData": {
      "RequestDetails": {
          "endpointType": "regional",
          "awsServingRegion": "us-east-1"
      }
  },
  "requestID": "aEXAMPLE-384d-46d2-b82e-a07f8EXAMPLE",
  "eventID": "aEXAMPLE-cd24-4544-b61b-fc668EXAMPLE",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "123456789012",
  "eventCategory": "Management",
  "tlsDetails": {
      "tlsVersion": "TLSv1.3",
      "cipherSuite": "TLS_AES_128_GCM_SHA256",
      "clientProvidedHostHeader": "sts.us-east-1.amazonaws.com"
  }
}
```


Example: Role assumption in the destination account

The following example shows the corresponding `AssumeRoleWithWebIdentity` event. Its `requestParameters.webIdentityTokenId` value matches the token identifier in the source event.


```
{
  "eventVersion": "1.11",
```

```
  "userIdentity": {
      "type": "WebIdentityUser",
      "principalId": "arn:aws-cn:iam::111122223333:oidc-provider/
a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:your-
audience:arn:aws:iam::123456789012:role/SourceWorkloadRole",
      "userName": "arn:aws:iam::123456789012:role/SourceWorkloadRole",
      "identityProvider": "arn:aws-cn:iam::111122223333:oidc-provider/
a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws"
  },
  "eventTime": "2026-10-02T01:05:06Z",
  "eventSource": "sts.amazonaws.com",
  "eventName": "AssumeRoleWithWebIdentity",
  "awsRegion": "cn-north-1",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "aws-cli/2.34.4 md/command#sts.assume-role-with-web-identity",
  "requestParameters": {
      "roleArn": "arn:aws-cn:iam::111122223333:role/DestinationWorkloadRole",
      "roleSessionName": "cross-partition-session",
      "webIdentityTokenId": "6e70a1d6-7cbd-44f5-9b94-d0540EXAMPLE"
  },
  "responseElements": {
      "credentials": {
          "accessKeyId": "ASIAI44QH8DHBEXAMPLE",
          "sessionToken": "<encoded session token blob>",
          "expiration": "2026-10-02T02:05:06Z"
      },
      "subjectFromWebIdentityToken": "arn:aws:iam::123456789012:role/
SourceWorkloadRole",
      "assumedRoleUser": {
          "assumedRoleId": "AROA987654321EXAMPLE:cross-partition-session",
          "arn": "arn:aws-cn:sts::111122223333:assumed-role/DestinationWorkloadRole/
cross-partition-session"
      },
      "provider": "arn:aws-cn:iam::111122223333:oidc-provider/
a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws",
      "audience": "your-audience",
      "packedPolicySize": 31,
      "sessionTokenSize": 1296,
      "sessionTokenUtilization": 31
  },
  "additionalEventData": {
      "RequestDetails": {
          "endpointType": "regional",
          "awsServingRegion": "cn-north-1"
```

```
        },
        "crossPartitionData": {
            "sourcePrincipalArn": "arn:aws:iam::123456789012:role/SourceWorkloadRole",
            "sourceRegion": "us-east-1"
        },
        "identityProviderConnectionVerificationMethod": "IAMTrustStore"
    },
    "requestID": "aEXAMPLE-bc02-4669-a0ea-dd53fEXAMPLE",
    "eventID": "aEXAMPLE-99a1-4df4-b279-5c647EXAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::IAM::Role",
            "ARN": "arn:aws-cn:iam::111122223333:role/DestinationWorkloadRole"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "sts.cn-north-1.amazonaws.com.cn"
    }
}
```

Temporary security credentials in IAM

You can use the AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. Temporary security credentials work almost identically to long-term access key credentials, with the following differences:

- Temporary security credentials are *short-term*, as the name implies. They can be configured to last for anywhere from a few minutes to several hours. After the credentials expire, AWS no longer recognizes them or allows any kind of access from API requests made with them.
- Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested. When (or even before) the temporary security credentials

expire, the user can request new credentials, as long as the user requesting them still has permissions to do so.

As a result, temporary credentials have the following advantages over long-term credentials:

- You do not have to distribute or embed long-term AWS security credentials with an application.
- You can provide access to your AWS resources to users without having to define an AWS identity for them. Temporary credentials are the basis for roles and identity federation.
- The temporary security credentials have a limited lifetime, so you do not have to update them or explicitly revoke them when they're no longer needed. After temporary security credentials expire, they cannot be reused. You can specify how long the credentials are valid, up to a maximum limit.

AWS STS and AWS regions

Temporary security credentials are generated by AWS STS. By default, AWS STS is a global service with a single endpoint at `https://sts.amazonaws.com`. However, you can also choose to make AWS STS API calls to endpoints in any other supported Region. This can reduce latency (server lag) by sending the requests to servers in a Region that is geographically closer to you. No matter which Region your credentials come from, they work globally. For more information, see Manage AWS STS in an AWS Region.

Common scenarios for temporary credentials

Temporary credentials are useful in scenarios that involve identity federation, delegation, cross-account access, and IAM roles.

Identity federation

You can manage your user identities in an external system outside of AWS and grant users who sign in from those systems access to perform AWS tasks and access your AWS resources. IAM supports two types of identity federation. In both cases, the identities are stored outside of AWS. The distinction is where the external system resides—in your data center or an external third party on the web. To compare features of temporary security credentials for identity federation, see Compare AWS STS credentials.

For more information about external identity providers, see Identity providers and federation into AWS.

- **OpenID Connect (OIDC) federation** – You can let users sign in using a well-known third-party identity provider such as Login with Amazon, Facebook, Google, or any OIDC-compatible provider. For your mobile or web application, you don't need to create custom sign-in code or manage your own user identities. Using OIDC federation helps you keep your AWS account secure, because you don't have to distribute long-term security credentials, such as IAM user access keys, with your application. For more information, see OIDC federation.

  AWS STS OIDC federation supports Login with Amazon, Facebook, Google, and any OpenID Connect (OIDC)-compatible identity provider.

  > **Note**
  >
  > For mobile applications, we recommend that you use Amazon Cognito. You can use this service with AWS SDKs for mobile development to create unique identities for users and authenticate them for secure access to your AWS resources. Amazon Cognito supports the same identity providers as AWS STS, and also supports unauthenticated (guest) access and lets you migrate user data when a user signs in. Amazon Cognito also provides API operations for synchronizing user data so that it is preserved as users move between devices. For more information, see Authentication with Amplify in the *Amplify Documentation*.

- **SAML federation** – You can authenticate users in your organization's network, and then provide those users access to AWS without creating new AWS identities for them and requiring them to sign in with different sign-in credentials. This is known as the *single sign-on* approach to temporary access. AWS STS supports open standards like Security Assertion Markup Language (SAML) 2.0, with which you can use Microsoft AD FS to leverage your Microsoft Active Directory. You can also use SAML 2.0 to manage your own solution for federating user identities. For more information, see SAML 2.0 federation.
  - **Custom federation broker** – You can use your organization's authentication system to grant access to AWS resources. For an example scenario, see Enable custom identity broker access to the AWS console.
  - **Federation using SAML 2.0** – You can use your organization's authentication system and SAML to grant access to AWS resources. For more information and an example scenario, see SAML 2.0 federation.

Roles for cross-account access

Many organizations maintain more than one AWS account. Using roles and cross-account access, you can define user identities in one account, and use those identities to access AWS resources in other accounts that belong to your organization. This is known as the *delegation* approach to temporary access. For more information about creating cross-account roles, see Create a role to give permissions to an IAM user. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

Roles for Amazon EC2

If you run applications on Amazon EC2 instances and those applications need access to AWS resources, you can provide temporary security credentials to your instances when you launch them. These temporary security credentials are available to all applications that run on the instance, so you don't need to store any long-term credentials on the instance. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances.

To learn more about IAM Amazon EC2 role credentials, see IAM roles for Amazon EC2 in the *Amazon Elastic Compute Cloud User Guide*.

Other AWS services

You can use temporary security credentials to access most AWS services. For a list of the services that accept temporary security credentials, see AWS services that work with IAM.

Sample applications that use temporary credentials

You can use AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see Temporary security credentials in IAM. To see how you can use AWS STS to manage temporary security credentials, you can download the following sample applications that implement complete example scenarios:

- Enabling Federation to AWS Using Windows Active Directory, ADFS, and SAML 2.0. Demonstrates how to delegate access using enterprise federation to AWS using Windows Active Directory (AD), Active Directory Federation Services (ADFS) 2.0, and SAML (Security Assertion Markup Language) 2.0.

- Enable custom identity broker access to the AWS console. Demonstrates how to create a custom federation proxy that enables single sign-on (SSO) so that existing Active Directory users can sign in to the AWS Management Console.
- How to Use Shibboleth for Single Sign-On to the AWS Management Console.. Shows how to use Shibboleth and SAML to provide users with single sign-on (SSO) access to the AWS Management Console.

Samples for OIDC federation

The following sample applications illustrate how to use OIDC federation with providers like Login with Amazon, Amazon Cognito, Facebook, or Google. You can trade authentication from these providers for temporary AWS security credentials to access AWS services.

- Amazon Cognito Tutorials – We recommend that you use Amazon Cognito with the AWS SDKs for mobile development. Amazon Cognito is the simplest way to manage identity for mobile apps, and it provides additional features like synchronization and cross-device identity. For more information about Amazon Cognito, see Authentication with Amplify in the *Amplify Documentation*.

Additional resources for temporary security credentials

The following scenarios and applications can guide you in using temporary security credentials:

- How to integrate AWS STS SourceIdentity with your identity provider. This post shows you how to set up the AWS STS `SourceIdentity` attribute when using Okta, Ping, or OneLogin as your IdP.
- OIDC federation. This section discusses how to configure IAM roles when you use OIDC federation and the `AssumeRoleWithWebIdentity` API.
- Secure API access with MFA. This topic explains how to use roles to require multi-factor authentication (MFA) to protect sensitive API actions in your account.

For more information on policies and permissions in AWS see the following topics:

- Access management for AWS resources
- Policy evaluation logic.

- Managing Access Permissions to Your Amazon S3 Resources in *Amazon Simple Storage Service User Guide*.
- To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

Compare AWS STS credentials

The following table compares features of the API operations in AWS STS that return temporary security credentials. To learn about the different methods you can use to request temporary security credentials by assuming a role, see Methods to assume a role. To learn about the different AWS STS API operations that allow you to pass session tags, see Pass session tags in AWS STS.

> **Note**
>
> You can send AWS STS API calls either to a global endpoint or to one of the Regional endpoints. If you choose an endpoint closer to you, you can reduce latency and improve the performance of your API calls. You also can choose to direct your calls to an alternative Regional endpoint if you can no longer communicate with the original endpoint. If you are using one of the various AWS SDKs, then use that SDK method to specify a Region before you make the API call. If you manually construct HTTP API requests, then you must direct the request to the correct endpoint yourself. For more information, see the AWS STS section of *Regions and Endpoints* and Manage AWS STS in an AWS Region.

AWS STS API	Who can call	Credential lifetime (min | max | default)	MFA support[1]	Session policy support[2]	Restrictions on resulting temporary credentials
AssumeRole	IAM user or IAM role with existing temporary security credentials	15 m | Maximum session duration	Yes	Yes	Cannot call `GetFederationToken` or `GetSessionToken`.

AWS STS API	Who can call	Credential lifetime (min | max | default)	MFA support[1]	Session policy support[2]	Restrictions on resulting temporary credentials
		setting[3] | 1 hr			
AssumeRoleWithSAML	Any user; caller must pass a SAML authentication response that indicates authentication from a known identity provider	15 m | Maximum session duration setting[3] | 1 hr	No	Yes	Cannot call `GetFederationToken` or `GetSessionToken`.
AssumeRoleWithWebIdentity	Any user; caller must pass an OIDC compliant JWT token that indicates authentication from a known identity provider	15 m | Maximum session duration setting[3] | 1 hr	No	Yes	Cannot call `GetFederationToken` or `GetSessionToken`.
GetFederationToken	IAM user or AWS account root user	IAM user: 15 m | 36 hr | 12 hr Root user: 15 m | 1 hr | 1 hr	No	Yes	Cannot call IAM operations using the AWS CLI or AWS API. This limitation does not apply to console sessions. Cannot call AWS STS operations except `GetCallerIdentity`.[4] SSO to console is allowed.[5]

AWS STS API	Who can call	Credential lifetime (min | max | default)	MFA support[1]	Session policy support[2]	Restrictions on resulting temporary credentials
GetSessionToken	IAM user or AWS account root user	IAM user: 15 m | 36 hr | 12 hr Root user: 15 m | 1 hr | 1 hr	Yes	No	Cannot call IAM API operations unless MFA information is included with the request. Cannot call AWS STS API operations except `AssumeRole` or `GetCallerIdentity` . SSO to console is not allowed.[6]

[1] **MFA support**. You can include information about a multi-factor authentication (MFA) device when you call the AssumeRole and GetSessionToken API operations. This ensures that the temporary security credentials that result from the API call can be used only by users who are authenticated with an MFA device. For more information, see Secure API access with MFA.

[2] **Session policy support**. Session policies are policies that you pass as a parameter when you programmatically create a temporary session for a role or AWS STS federated user session. This policy limits the permissions from the role or user's identity-based policy that are assigned to the session. The resulting session's permissions are the intersection of the entity's identity-based policies and the session policies. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. For more information about role session permissions, see Session policies.

[3] **Maximum session duration setting**. Use the `DurationSeconds` parameter to specify the duration of your role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role. To learn how to view the maximum value for your role, see Update the maximum session duration for a role.

[4] **GetCallerIdentity**. No permissions are required to perform this operation. If an administrator adds a policy to your IAM user or role that explicitly denies access to the `sts:GetCallerIdentity` action, you can still perform this operation. Permissions are not required because the same information is returned when an IAM user or role is denied access. To view an example response, see I am not authorized to perform: iam:DeleteVirtualMFADevice.

[5] **Single sign-on (SSO) to the console**. To support SSO, AWS lets you call a federation endpoint (`https://signin.aws.amazon.com/federation`) and pass temporary security credentials. The endpoint returns a token that you can use to construct a URL that signs a user directly into the console without requiring a password. For more information, see Enabling SAML 2.0 federated principals to access the AWS Management Console and How to Enable Cross-Account Access to the AWS Management Console in the AWS Security Blog.

[6] After you retrieve your temporary credentials, you can't access the AWS Management Console by passing the credentials to the federation single sign-on endpoint. For more information, see Enable custom identity broker access to the AWS console.

Service bearer tokens

Some AWS services require that you have permission to get an AWS STS service bearer token before you can access their resources programmatically. These services support a protocol that requires you to use a bearer token instead of using a traditional AWS Signature Version 4 for API requests. When you perform AWS CLI or AWS API operations that require bearer tokens, the AWS service requests a bearer token on your behalf. The service provides you with the token, which you can then use to perform subsequent operations in that service.

AWS STS service bearer tokens include information from your original principal authentication that might affect your permissions. This information can include principal tags, session tags, and session policies. The token's access key ID begins with the `ABIA` prefix. This helps you to identify operations that were performed using service bearer tokens in your CloudTrail logs.

> **Important**
>
> The bearer token can be used only for calls to the service that generates it and in the Region where it was generated. You can't use the bearer token to perform operations in other services or Regions.

An example of a service that supports bearer tokens is AWS CodeArtifact. Before you can interact with AWS CodeArtifact using a package manager such as NPM, Maven, or PIP, you must call the `aws codeartifact get-authorization-token` operation. This operation returns a bearer token that you can use to perform AWS CodeArtifact operations. Alternatively, you can use the `aws codeartifact login` command that completes the same operation and then configures your client automatically.

If you perform an action in an AWS service that generates a bearer token for you, you must have the following permissions in your IAM policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowServiceBearerToken",
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*"
        }
    ]
}
```

For a service bearer token example, see Using identity-based policies for AWS CodeArtifact in the *AWS CodeArtifact* user guide.

Request temporary security credentials

To request temporary security credentials, you can use AWS Security Token Service (AWS STS) operations in the AWS API. These include operations to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see Temporary security credentials in IAM. To learn about the different methods that you can use to request temporary security credentials by assuming a role, see Methods to assume a role.

To call the API operations, you can use one of the AWS SDKs. The SDKs are available for a variety of programming languages and environments, including Java, .NET, Python, Ruby, Android,

and iOS. The SDKs take care of tasks such as cryptographically signing your requests, retrying requests if necessary, and handling error responses. You can also use the AWS STS Query API, which is described in the AWS Security Token Service API Reference. Finally, two command line tools support the AWS STS commands: the AWS Command Line Interface, and the AWS Tools for Windows PowerShell.

The AWS STS API operations create a new session with temporary security credentials that include an access key pair and a session token. The access key pair consists of an access key ID and a secret key. Users (or an application that the user runs) can use these credentials to access your resources.

You can create a role session and pass session policies and session tags programmatically using AWS STS API operations. The resulting session permissions are the intersection of the role's identity-based policies and the session policies. For more information about session policies, see Session policies. For more information about session tags, see Pass session tags in AWS STS.

> **Note**
>
> The size of the session token that AWS STS API operations return is not fixed. We strongly recommend that you make no assumptions about the maximum size. The typical token size is less than 4096 bytes, but that can vary.

Using AWS STS with AWS Regions

You can send AWS STS API calls either to a global endpoint or to one of the Regional endpoints. If you choose an endpoint closer to you, you can reduce latency and improve the performance of your API calls. You also can choose to direct your calls to an alternative Regional endpoint if you can no longer communicate with the original endpoint.

If you are using one of the various AWS SDKs, then use that SDK method to specify a Region before you make the API call. If you manually construct HTTP API requests, then you must direct the request to the correct endpoint yourself. For more information, see the AWS STS section of *Regions and Endpoints* and Manage AWS STS in an AWS Region.

The following are the API operations that you can use to acquire temporary credentials for use in your AWS environment and applications.

Requesting credentials for cross-account delegation and federation through a custom identity broker

The `AssumeRole` API operation is useful for allowing existing IAM users to access AWS resources that they don't already have access to. For example, the user might need access to resources in another AWS account. It is also useful as a means to temporarily gain privileged access—for example, to provide multi-factor authentication (MFA). You must call this API using active credentials. To learn who can call this operation, see Compare AWS STS credentials. For more information, see Create a role to give permissions to an IAM user and Secure API access with MFA.

To request temporary security credentials for cross-account delegation and federation through a custom identity broker

1. Authenticate with your AWS security credentials. This call must be made using valid AWS security credentials.
2. Call the operation `AssumeRole`.

The following example shows a sample request and response using `AssumeRole`. This example request assumes the `demo` role for the specified duration with the included session policy, session tags, external ID, and source identity. The resulting session is named `John-session`.

Example request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=AssumeRole
&RoleSessionName=John-session
&RoleArn=arn:aws:iam::123456789012:role/demo
&Policy=%7B%22Version%22%3A%222012-10-17       %22%2C%22Statement%22%3A%5B%7B%22Sid
%22%3A%20%22Stmt1%22%2C%22Effect%22%3A%20%22Allow%22%2C%22Action%22%3A%20%22s3%3A*
%22%2C%22Resource%22%3A%20%22*%22%7D%5D%7D
&DurationSeconds=1800
&Tags.member.1.Key=Project
&Tags.member.1.Value=Pegasus
&Tags.member.2.Key=Cost-Center
&Tags.member.2.Value=12345
&ExternalId=123ABC
&SourceIdentity=DevUser123
&AUTHPARAMS
```

The policy value shown in the preceding example is the URL-encoded version of the following policy:

JSON

```
{"Version":"2012-10-17",      "Statement":
[{"Sid":"Stmt1","Effect":"Allow","Action":"s3:*","Resource":"*"}]}
```

The AUTHPARAMS parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.

In addition to the temporary security credentials, the response includes the Amazon Resource Name (ARN) for the federated user and the expiration time of the credentials.

Example response

```
<AssumeRoleResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<AssumeRoleResult>
<SourceIdentity>DevUser123</SourceIdentity>
<Credentials>
  <SessionToken>
   AQoDYXdzEPT//////////wEXAMPLEtc764bNrC9SAPBSM22wDOk4x4HIZ8j4FZTwdQW
   LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTflfKD8YUuwthAx7mSEI/qkPpKPi/kMcGd
   QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSXDvp75YU
   9HFvlRd8Tx6q6fE8YQcHNVXAkiY9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL64lIZbqBAz
   +scqKmlzm8FDrypNC9Yjc8fPOLn9FX9KSYvKTr4rvx3iSIlTJabIQwj2ICCR/oLxBA==
  </SessionToken>
  <SecretAccessKey>
   wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
  </SecretAccessKey>
  <Expiration>2019-07-15T23:28:33.359Z</Expiration>
  <AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
</Credentials>
<AssumedRoleUser>
  <Arn>arn:aws:sts::123456789012:assumed-role/demo/John</Arn>
  <AssumedRoleId>ARO123EXAMPLE123:John</AssumedRoleId>
```

```
</AssumedRoleUser>
<PackedPolicySize>34</PackedPolicySize>
<SessionTokenSize>1428</SessionTokenSize>
<SessionTokenUtilization>34</SessionTokenUtilization>
</AssumeRoleResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</AssumeRoleResponse>
```

Note

Session policies and session tags add to the session token size. If the token exceeds the size limit, your request fails, even if your plaintext is within the other limits. The `SessionTokenUtilization` response element shows what percentage of the size limit the token uses. The `PackedPolicySize` response element is deprecated. Use `SessionTokenUtilization` instead.

Requesting credentials through an OIDC provider

The `AssumeRoleWithWebIdentity` API operation returns a set of temporary AWS security credentials in exchange for a JSON Web Token (JWT). This includes public identity providers, such as Login with Amazon, Facebook, Google, and providers that issue JWTs that are compatible with OpenID Connect (OIDC) discovery, such as GitHub actions or Azure Devops. For more information, see OIDC federation.

Note

`AssumeRoleWithWebIdentity` requests are not signed with, and do not require AWS credentials.

Requesting credentials through an OIDC provider

1. Call the operation `AssumeRoleWithWebIdentity`.

   When you call `AssumeRoleWithWebIdentity`, AWS validates the token presented by verifying the digital signature using public keys made available through your IdP's JSON web

keyset (JWKS). If the token is valid, and all conditions set forth in the IAM role trust policy are met, AWS returns the following information to you:

- A set of temporary security credentials. These consist of an access key ID, a secret access key, and a session token.
- The role ID and the ARN of the assumed role.
- A `SubjectFromWebIdentityToken` value that contains the unique user ID.

2. Your application may then use the temporary security credentials that were returned in the response to make AWS API calls. This is the same process as making an AWS API call with long-term security credentials. The difference is that you must include the session token, which lets AWS verify that the temporary security credentials are valid.

Your application should cache the credentials returned by AWS STS and refresh them as needed. If your application is built using an AWS SDK, the SDK has credential providers that can handle calling `AssumeRoleWithWebIdentity` and refreshing AWS credentials before they expire. For more information, see AWS SDKs and Tools standardized credential providers in the *AWS SDKs and Tools Reference Guide*.

Requesting credentials through a SAML 2.0 identity provider

The `AssumeRoleWithSAML` API operation returns a set of temporary security credentials for SAML federated principals who are authenticated by your organization's existing identity system. The users must also use SAML 2.0 (Security Assertion Markup Language) to pass authentication and authorization information to AWS. This API operation is useful in organizations that have integrated their identity systems (such as Windows Active Directory or OpenLDAP) with software that can produce SAML assertions. Such an integration provides information about user identity and permissions (such as Active Directory Federation Services or Shibboleth). For more information, see SAML 2.0 federation.

1. Call the operation `AssumeRoleWithSAML`.

   This is an unsigned call, meaning you do not need to authenticate AWS security credentials prior to making the request.

> **Note**
>
> A call to `AssumeRoleWithSAML` is not signed (encrypted). Therefore, you should only include optional session policies if the request is transmitted through a trusted intermediary. In this case, someone could alter the policy to remove the restrictions.

2. When you call `AssumeRoleWithSAML`, AWS verifies the authenticity of the SAML assertion. Assuming that the identity provider validates the assertion, AWS returns the following information to you:

   - A set of temporary security credentials. These consist of an access key ID, a secret access key, and a session token.
   - The role ID and the ARN of the assumed role.
   - An `Audience` value that contains the value of the `Recipient` attribute of the `SubjectConfirmationData` element of the SAML assertion.
   - An `Issuer` value that contains the value of the `Issuer` element of the SAML assertion.
   - A `NameQualifier` element that contains a hash value built from the `Issuer` value, the AWS account ID, and the friendly name of the SAML provider. When combined with the `Subject` element, they can uniquely identify the SAML federated principal.
   - A `Subject` element that contains the value of the `NameID` element in the `Subject` element of the SAML assertion.
   - A `SubjectType` element that indicates the format of the `Subject` element. The value can be `persistent`, `transient`, or the full `Format` URI from the `Subject` and `NameID` elements used in your SAML assertion. For information about the `NameID` element's `Format` attribute, see Configure SAML assertions for the authentication response.

3. Use the temporary security credentials returned in the response to make AWS API calls. This is the same process as making an AWS API call with long-term security credentials. The difference is that you must include the session token, which lets AWS verify that the temporary security credentials are valid.

Your app should cache the credentials. By default the credentials expire after an hour. If you are not using the AmazonSTSCredentialsProvider action in the AWS SDK, it's up to you and your app to call `AssumeRoleWithSAML` again. Call this operation to get a new set of temporary security credentials before the old ones expire.

Requesting credentials through a custom identity broker

The `GetFederationToken` API operation returns a set of temporary security credentials for AWS STS federated user principals. This API differs from `AssumeRole` in that the default expiration period is substantially longer (12 hours instead of one hour). Additionally, you can use the `DurationSeconds` parameter to specify a duration for the temporary security credentials to remain valid. The resulting credentials are valid for the specified duration, between 900 seconds (15 minutes) to 129,600 seconds (36 hours). The longer expiration period can help reduce the number of calls to AWS because you do not need to get new credentials as often.

1. Authenticate with the AWS security credentials of your specific IAM user. This call must be made using valid AWS security credentials.
2. Call the operation `GetFederationToken`.

The `GetFederationToken` call returns temporary security credentials that consist of the session token, access key, secret key, and expiration. You can use `GetFederationToken` if you want to manage permissions inside your organization (for example, using the proxy application to assign permissions).

The following example shows a sample request and response that uses `GetFederationToken`. This example request federates the calling user for the specified duration with the session policy ARN and session tags. The resulting session is named `Jane-session`.

Example request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetFederationToken
&Name=Jane-session
&PolicyArns.member.1.arn==arn%3Aaws%3Aiam%3A%3A123456789012%3Apolicy%2FRole1policy
&DurationSeconds=1800
&Tags.member.1.Key=Project
&Tags.member.1.Value=Pegasus
&Tags.member.2.Key=Cost-Center
&Tags.member.2.Value=12345
&AUTHPARAMS
```

The policy ARN shown in the preceding example includes the following URL-encoded ARN:

`arn:aws:iam::123456789012:policy/Role1policy`

Also, note that the &AUTHPARAMS parameter in the example is meant as a placeholder for the authentication information. This is the *signature*, which you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.

In addition to the temporary security credentials, the response includes the Amazon Resource Name (ARN) for the federated user and the expiration time of the credentials.

Example response

```
<GetFederationTokenResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<GetFederationTokenResult>
<Credentials>
  <SessionToken>
   AQoDYXdzEPT//////////wEXAMPLEtc764bNrC9SAPBSM22wDOk4x4HIZ8j4FZTwdQW
   LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTflfKD8YUuwthAx7mSEI/qkPpKPi/kMcGd
   QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSXDvp75YU
   9HFvlRd8Tx6q6fE8YQcHNVXAkiY9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL64lIZbqBAz
   +scqKmlzm8FDrypNC9Yjc8fPOLn9FX9KSYvKTr4rvx3iSIlTJabIQwj2ICCEXAMPLE==
  </SessionToken>
  <SecretAccessKey>
  wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
  </SecretAccessKey>
  <Expiration>2019-04-15T23:28:33.359Z</Expiration>
  <AccessKeyId>AKIAIOSFODNN7EXAMPLE;</AccessKeyId>
</Credentials>
<FederatedUser>
  <Arn>arn:aws:sts::123456789012:federated-user/Jean</Arn>
  <FederatedUserId>123456789012:Jean</FederatedUserId>
</FederatedUser>
<PackedPolicySize>26</PackedPolicySize>
<SessionTokenSize>1105</SessionTokenSize>
<SessionTokenUtilization>26</SessionTokenUtilization>
</GetFederationTokenResult>
<ResponseMetadata>
<RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
</ResponseMetadata>
</GetFederationTokenResponse>
```

> **Note**
>
> Session policies and session tags add to the session token size. If the token exceeds the size limit, your request fails, even if your plaintext is within the other limits. The `SessionTokenUtilization` response element shows what percentage of the size limit the token uses. The `PackedPolicySize` response element is deprecated. Use `SessionTokenUtilization` instead.

AWS recommends that you grant permissions at the resource level (for example, you attach a resource-based policy to an Amazon S3 bucket), you can omit the `Policy` parameter. However, if you do not include a policy for the AWS STS federated user principal, the temporary security credentials will not grant any permissions. In this case, you *must* use resource policies to grant the federated user access to your AWS resources.

For example, assume your AWS account number is 111122223333, and you have an Amazon S3 bucket that you want to allow Susan to access. Susan's temporary security credentials don't include a policy for the bucket. In that case, you would need to ensure that the bucket has a policy with an ARN that matches Susan's ARN, such as `arn:aws:sts::111122223333:federated-user/Susan`.

Requesting credentials for users in untrusted environments

The `GetSessionToken` API operation returns a set of temporary security credentials to an existing IAM user. This is useful for providing enhanced security, such as allowing AWS requests only when MFA is enabled for the IAM user. Because the credentials are temporary, they provide enhanced security when you have an IAM user who accesses your resources through a less secure environment. Examples of less secure environments include a mobile device or web browser.

1. Authenticate with the AWS security credentials of your specific IAM user. This call must be made using valid AWS security credentials.
2. Call the operation `GetSessionToken`.
3. `GetSessionToken` returns temporary security credentials consisting of a session token, an access key ID, and a secret access key.

By default, temporary security credentials for an IAM user are valid for a maximum of 12 hours. But you can request a duration as short as 15 minutes or as long as 36 hours using the

`DurationSeconds` parameter. For security reasons, a token for an AWS account root user is restricted to a duration of one hour.

The following example shows a sample request and response using `GetSessionToken`. The response also includes the expiration time of the temporary security credentials.

Example request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=1800
&AUTHPARAMS
```

The AUTHPARAMS parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.

Example response

```
<GetSessionTokenResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
<GetSessionTokenResult>
<Credentials>
  <SessionToken>
   AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrRh3c/L
   To6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3z
   rkuWJOgQs8IZZaIv2BXIa2R4OlgkBN9bkUDNCJiBeb/AXlzBBko7b15fjrBs2+cTQtp
   Z3CYWFXG8C5zqx37wnOE49mRl/+OtkIKGO7fAE
  </SessionToken>
  <SecretAccessKey>
  wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
  </SecretAccessKey>
  <Expiration>2011-07-11T19:55:29.611Z</Expiration>
  <AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
</Credentials>
<SessionTokenSize>856</SessionTokenSize>
<SessionTokenUtilization>20</SessionTokenUtilization>
</GetSessionTokenResult>
```

```
<ResponseMetadata>
<RequestId>58c5dbae-abef-11e0-8cfe-09039844ac7d</RequestId>
</ResponseMetadata>
</GetSessionTokenResponse>
```

Optionally, the `GetSessionToken` request can include `SerialNumber` and `TokenCode` values for AWS multi-factor authentication (MFA) verification. If the provided values are valid, AWS STS provides temporary security credentials that include the state of MFA authentication. The temporary security credentials can then be used to access the MFA-protected API operations or AWS websites for as long as the MFA authentication is valid.

The following example shows a `GetSessionToken` request that includes an MFA verification code and device serial number.

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=7200
&SerialNumber=YourMFADeviceSerialNumber
&TokenCode=123456
&AUTHPARAMS
```

Note

The call to AWS STS can be to the global endpoint or to any of the Regional endpoints that you activate your AWS account. For more information, see the AWS STS section of *Regions and Endpoints*.

The `AUTHPARAMS` parameter in the example is a placeholder for your *signature*. A signature is the authentication information that you must include with AWS HTTP API requests. We recommend using the AWS SDKs to create API requests, and one benefit of doing so is that the SDKs handle request signing for you. If you must create and sign API requests manually, see Signing AWS Requests By Using Signature Version 4 in the *Amazon Web Services General Reference* to learn how to sign a request.

Use temporary credentials with AWS resources

You can use temporary security credentials to make programmatic requests for AWS resources using the AWS CLI or AWS API (using the AWS SDKs). The temporary credentials provide the same

permissions as long-term security credentials, such as IAM user credentials. However, there are a few differences:

- When you make a call using temporary security credentials, the call must include a session token, which is returned along with those temporary credentials. AWS uses the session token to validate the temporary security credentials.
- Temporary credentials expire after a specified interval. After temporary credentials expire, any calls that you make with those credentials will fail, so you must generate a new set of temporary credentials. Temporary credentials cannot be extended or refreshed beyond the original specified interval.
- When you use temporary credentials to make a request, your principal might include a set of tags. These tags come from session tags and tags that are attached to the role that you assume. For more information about session tags, see Pass session tags in AWS STS.

If you are using the AWS SDKs, the AWS Command Line Interface (AWS CLI), or the Tools for Windows PowerShell, the way to get and use temporary security credentials differs with the context. If you are running code, AWS CLI, or Tools for Windows PowerShell commands inside an EC2 instance, you can take advantage of roles for Amazon EC2. Otherwise, you can call an AWS STS API to get the temporary credentials, and then use them explicitly to make calls to AWS services.

> **Note**
>
> You can use AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see Temporary security credentials in IAM. AWS STS is a global service that has a default endpoint at `https://sts.amazonaws.com`. This endpoint is in the US East (N. Virginia) Region, although credentials that you get from this and other endpoints are valid globally. These credentials work with services and resources in any Region. You can also choose to make AWS STS API calls to endpoints in any of the supported Regions. This can reduce latency by making the requests from servers in a Region that is geographically closer to you. No matter which Region your credentials come from, they work globally. For more information, see Manage AWS STS in an AWS Region.

Contents

- Using temporary credentials in Amazon EC2 instances

- Using temporary security credentials with the AWS SDKs
- Using temporary security credentials with the AWS CLI
- Using temporary security credentials with API operations
- More information

Using temporary credentials in Amazon EC2 instances

If you want to run AWS CLI commands or code inside an EC2 instance, the recommended way to get credentials is to use roles for Amazon EC2. You create an IAM role that specifies the permissions that you want to grant to applications that run on the EC2 instances. When you launch the instance, you associate the role with the instance.

Applications, AWS CLI, and Tools for Windows PowerShell commands that run on the instance can then get automatic temporary security credentials from the instance metadata. You do not have to explicitly get the temporary security credentials. The AWS SDKs, AWS CLI, and Tools for Windows PowerShell automatically get the credentials from the EC2 Instance Metadata Service (IMDS) and use them. The temporary credentials have the permissions that you define for the role that is associated with the instance.

For more information and for examples, see the following:

- Using IAM Roles to Grant Access to AWS Resources on Amazon Elastic Compute Cloud — AWS SDK for Java
- Granting Access Using an IAM Role — AWS SDK for .NET
- Creating a Role — AWS SDK for Ruby

Using temporary security credentials with the AWS SDKs

To use temporary security credentials in code, you programmatically call an AWS STS API like `AssumeRole` and extract the resulting credentials and session token. You then use those values as credentials for subsequent calls to AWS. The following example shows pseudocode for how to use temporary security credentials if you're using an AWS SDK:

```
assumeRoleResult = AssumeRole(role-arn);
tempCredentials = new SessionAWSCredentials(
   assumeRoleResult.AccessKeyId,
```

```
   assumeRoleResult.SecretAccessKey,
   assumeRoleResult.SessionToken);
s3Request = CreateAmazonS3Client(tempCredentials);
```

For an example written in Python (using the AWS SDK for Python (Boto)), see Switch to an IAM role (AWS API). This example shows how to call `AssumeRole` to get temporary security credentials and then use those credentials to make a call to Amazon S3.

For details about how to call `AssumeRole`, `GetFederationToken`, and other API operations, see the AWS Security Token Service API Reference. For information on getting the temporary security credentials and session token from the result, see the documentation for the SDK that you're working with. You can find the documentation for all the AWS SDKs on the main AWS documentation page, in the **SDKs and Toolkits** section.

You must make sure that you get a new set of credentials before the old ones expire. In some SDKs, you can use a provider that manages the process of refreshing credentials for you; check the documentation for the SDK you're using.

Using temporary security credentials with the AWS CLI

You can use temporary security credentials with the AWS CLI. This can be useful for testing policies.

Using the AWS CLI, you can call an AWS STS API like `AssumeRole` or `GetFederationToken` and then capture the resulting output. The following example shows a call to `AssumeRole` that sends the output to a file. In the example, the `profile` parameter is assumed to be a profile in the AWS CLI configuration file. It is also assumed to reference credentials for an IAM user who has permissions to assume the role.

```
aws sts assume-role --role-arn arn:aws:iam::123456789012:role/role-name --role-session-
name "RoleSession1" --profile IAM-user-name > assume-role-output.txt
```

When the command is finished, you can extract the access key ID, secret access key, and session token from wherever you've routed it. You can do this either manually or by using a script. You can then assign these values to environment variables.

When you run AWS CLI commands, the AWS CLI looks for credentials in a specific order—first in environment variables and then in the configuration file. Therefore, after you've put the temporary credentials into environment variables, the AWS CLI uses those credentials by default. (If you specify a `profile` parameter in the command, the AWS CLI skips the environment variables.

Instead, the AWS CLI looks in the configuration file, which lets you override the credentials in the environment variables if you need to.)

The following example shows how you might set the environment variables for temporary security credentials and then call an AWS CLI command. Because no `profile` parameter is included in the AWS CLI command, the AWS CLI looks for credentials first in environment variables and therefore uses the temporary credentials.

Linux

```
$ export AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of session token>
$ aws ec2 describe-instances --region us-west-1
```

Windows

```
C:\> SET AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
C:\> SET AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
C:\> SET AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of token>
C:\> aws ec2 describe-instances --region us-west-1
```

Using temporary security credentials with API operations

If you're making direct HTTPS API requests to AWS, you can sign those requests with the temporary security credentials that you get from the AWS Security Token Service (AWS STS). To do this, you use the access key ID and secret access key that you receive from AWS STS. You use the access key ID and secret access key the same way you would use long-term credentials to sign a request. You also add to your API request the session token that you receive from AWS STS. You add the session token to an HTTP header or to a query string parameter named `X-Amz-Security-Token`. You add the session token to the HTTP header *or* the query string parameter, but not both. For more information about signing HTTPS API requests, see Signing AWS API Requests in the *AWS General Reference*.

More information

For more information about using AWS STS with other AWS services, see the following links:

- **Amazon S3**. See Making requests using IAM user temporary credentials or Making requests using federated user temporary credentials in the *Amazon Simple Storage Service User Guide*.

- **Amazon SNS**. See Using identity-based policies with Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.
- **Amazon SQS**. See Identity and access management in Amazon SQS in the *Amazon Simple Queue Service Developer Guide*.
- **Amazon SimpleDB**. See Using Temporary Security Credentials in the *Amazon SimpleDB Developer Guide*.

Permissions for temporary security credentials

You can use AWS Security Token Service (AWS STS) to create and provide trusted users with temporary security credentials that can control access to your AWS resources. For more information about AWS STS, see Temporary security credentials in IAM. After AWS STS issues temporary security credentials, they are valid through the expiration period and cannot be revoked. However, the permissions assigned to temporary security credentials are evaluated each time a request is made that uses the credentials, so you can achieve the effect of revoking the credentials by changing their access rights after they have been issued.

The following topics assume you have a working knowledge of AWS permissions and policies. For more information on these topics, see Access management for AWS resources.

Topics

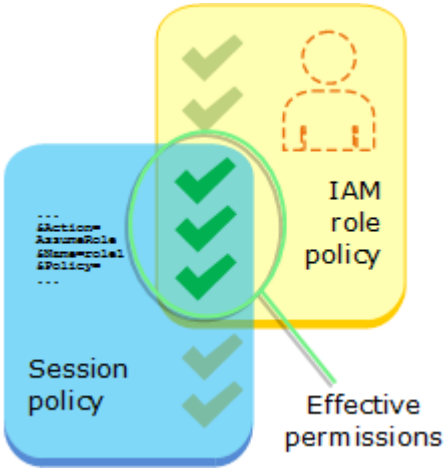
- Permissions for AssumeRole, AssumeRoleWithSAML, and AssumeRoleWithWebIdentity
- Monitor and control actions taken with assumed roles
- Permissions for GetFederationToken
- Permissions for GetSessionToken
- Disabling permissions for temporary security credentials
- Granting permissions to create temporary security credentials
- Granting permissions to use identity-enhanced console sessions

Permissions for AssumeRole, AssumeRoleWithSAML, and AssumeRoleWithWebIdentity

The permissions policy of the role that is being assumed determines the permissions for the temporary security credentials that are returned by `AssumeRole`, `AssumeRoleWithSAML`, and

`AssumeRoleWithWebIdentity`. You define these permissions when you create or update the role.

Optionally, you can pass inline or managed session policies as parameters of the `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity` API operations. Session policies limit the permissions for the role's temporary credential session. The resulting session's permissions are the intersection of the role's identity-based policy and the session policies. You can use the role's temporary credentials in subsequent AWS API calls to access resources in the account that owns the role. You cannot use session policies to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. To learn more about how AWS determines the effective permissions of a role, see Policy evaluation logic.



The policies that are attached to the credentials that made the original call to `AssumeRole` are not evaluated by AWS when making the "allow" or "deny" authorization decision. The user temporarily gives up its original permissions in favor of the permissions assigned by the assumed role. In the case of the `AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity` API operations, there are no policies to evaluate because the caller of the API is not an AWS identity.

Example: Assigning permissions using AssumeRole

You can use the `AssumeRole` API operation with different kinds of policies. Here are a few examples.

Role permissions policy

In this example, you call the `AssumeRole` API operation without specifying the session policy in the optional `Policy` parameter. The permissions assigned to the temporary credentials are determined by the permissions policy of the role being assumed. The following example

permissions policy grants the role permission to list all objects that are contained in an S3 bucket named `productionapp`. It also allows the role to get, put, and delete objects within that bucket.

Example role permissions policy

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::productionapp"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::productionapp/*"
    }
  ]
}
```

Session policy passed as a parameter

Imagine that you want to allow a user to assume the same role as in the previous example. But in this case you want the role session to have permission only to get and put objects in the `productionapp` S3 bucket. You do not want to allow them to delete objects. One way to accomplish this is to create a new role and specify the desired permissions in that role's permissions policy. Another way to accomplish this is to call the `AssumeRole` API and include session policies in the optional `Policy` parameter as part of the API operation. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Session policies cannot be used to grant more permissions than those allowed by the identity-based policy of the role that is being assumed. For more information about role session permissions, see Session policies.

After you retrieve the new session's temporary credentials, you can pass them to the user that you want to have those permissions.

For example, imagine that the following policy is passed as a parameter of the API call. The person using the session has permissions to perform only these actions:

- List all objects in the `productionapp` bucket.
- Get and put objects in the `productionapp` bucket.

In the following session policy, the `s3:DeleteObject` permission is filtered out and the assumed session is not granted the `s3:DeleteObject` permission. The policy sets the maximum permissions for the role session so that it overrides any existing permissions policies on the role.

Example session policy passed with AssumeRole API call

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::productionapp"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::productionapp/*"
    }
  ]
}
```

Resource-based policy

Some AWS resources support resource-based policies, and these policies provide another mechanism to define permissions that affect temporary security credentials. Only a few resources,

like Amazon S3 buckets, Amazon SNS topics, and Amazon SQS queues support resource-based policies. The following example expands on the previous examples, using an S3 bucket named `productionapp`. The following policy is attached to the bucket.

When you attach the following resource-based policy to the `productionapp` bucket, *all* users are denied permission to delete objects from the bucket. (See the `Principal` element in the policy.) This includes all assumed role users, even though the role permissions policy grants the `DeleteObject` permission. An explicit `Deny` statement always takes precedence over an `Allow` statement.

Example bucket policy

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Principal": {"AWS": "*"},
    "Effect": "Deny",
    "Action": "s3:DeleteObject",
    "Resource": "arn:aws:s3:::productionapp/*"
  }
}
```

For more information about how multiple policy types are combined and evaluated by AWS, see Policy evaluation logic.

Monitor and control actions taken with assumed roles

An IAM role is an object in IAM that is assigned permissions. When you assume that role using an IAM identity or an identity from outside of AWS, you receive a session with the permissions that are assigned to the role.

When you perform actions in AWS, the information about your session can be logged to AWS CloudTrail for your account administrator to monitor. Administrators can configure roles to require identities to pass a custom string that identifies the person or application that is performing actions in AWS. This identity information is stored as the *source identity* in AWS CloudTrail. When the administrator reviews activity in CloudTrail, they can view the source identity information to determine who or what performed actions with assumed role sessions.

After a source identity is set, it is present in requests for any AWS action taken during the role session. The value that is set persists when a role is used to assume another role through the AWS CLI or AWS API, known as role chaining. The value that is set cannot be changed during the role session.

Administrators can configure granular permissions based on the presence or value of the source identity to further control AWS actions that are taken with shared roles. You can decide whether the source identity attribute can be used, whether it is required, and what value can be used.

The way that you use source identity differs from role session name and session tags in an important way. The source identity value can't be changed after it is set, and it persists for any additional actions that are taken with the role session. Here's how you can use session tags and role session name:

- **Session tags** – You can pass session tags when you assume a role or federate a user. Session tags are present when a role is assumed. You can define policies that use tag condition keys to grant permissions to your principals based on their tags. Then you can use CloudTrail to view the requests made to assume roles or federate users. To learn more about session tags, see Pass session tags in AWS STS.
- **Role session name** – You can use the `sts:RoleSessionName` condition key in a role trust policy to require that your users provide a specific session name when they assume a role. Role session name can be used to differentiate role sessions when a role is used by different principals. To learn more about role session name, see sts:RoleSessionName.

We recommend that you use source identity when you want to control the identity that assumes a role. Source identity is also useful for mining CloudTrail logs to determine who used the role to perform actions.

Topics

- Setting up to use source identity
- Things to know about source identity
- Permissions required to set source identity
- Specifying a source identity when assuming a role
- Using source identity with AssumeRole
- Using source identity with AssumeRoleWithSAML

- Using source identity with AssumeRoleWithWebIdentity
- Control access using source identity information
- Viewing source identity in CloudTrail

Setting up to use source identity

The way that you set up to use source identity depends on the method used when your roles are assumed. For example, your IAM users might assume roles directly using the `AssumeRole` operation. If you have enterprise identities, also known as workforce identities, they might access your AWS resources using `AssumeRoleWithSAML`. If end users access your mobile or web applications, they might do so using `AssumeRoleWithWebIdentity`. The following is a high-level workflow overview to help you understand how you can set up to utilize source identity information in your existing environment.

1. **Configure test users and roles** – Using a preproduction environment, configure test users and roles and configure their policies to allow setting a source identity.

   If you use an identity provider (IdP) for your federated identities, configure your IdP to pass a user attribute of your choice for source identity in the assertion or token.
2. **Assume the role** – Test assuming roles and passing a source identity with the users and roles that you set up for testing.
3. **Review CloudTrail** – Review the source identity information for your test roles in your CloudTrail logs.
4. **Train your users** – After you've tested in your preproduction environment, ensure that your users know how to pass in the source identity information, if necessary. Set a deadline for when you will require your users to provide a source identity in your production environment.
5. **Configure production policies** – Configure your policies for your production environment, and then add them to your production users and roles.
6. **Monitor activity** – Monitor your production role activity using CloudTrail logs.

Things to know about source identity

Keep the following in mind when working with source identity.

- Trust policies for all roles connected to an identity provider (IdP) must have the `sts:SetSourceIdentity` permission. For roles that don't have this permission in the role

trust policy, the `AssumeRole*` operation will fail. If you don't want to update the role trust policy for each role, you can use a separate IdP instance for passing source identity. Then add the `sts:SetSourceIdentity` permission to only the roles that are connected to the separate IdP.

- When an identity sets a source identity, the `sts:SourceIdentity` key is present in the request. For subsequent actions taken during the role session, the `aws:SourceIdentity` key is present in the request. AWS doesn't control the value of the source identity in either the `sts:SourceIdentity` or `aws:SourceIdentity` keys. If you choose to require a source identity, you must choose an attribute that you want your users or IdP to provide. For security purposes, you must ensure that you can control how those values are provided.
- The value of source identity must be between 2 and 256 characters long, can contain only alphanumeric characters, underscores, and the following characters: **. , + = @ -** (hyphen). You cannot use a value that begins with the text **`aws:`**. This prefix is reserved for AWS internal use.
- The source identity information is not captured by CloudTrail when an AWS service or service-linked role carries out an action on behalf of a federated or workforce identity.

> **Important**
>
> You cannot switch to a role in the AWS Management Console that requires a source identity to be set when the role is assumed. To assume such a role, you can use the AWS CLI or AWS API to call the `AssumeRole` operation and specify the source identity parameter.

Permissions required to set source identity

In addition to the action that matches the API operation, you must have the following permissions-only action in your policy:

```
sts:SetSourceIdentity
```

- To specify a source identity, principals (IAM users and roles) must have permissions to `sts:SetSourceIdentity`. As the administrator, you can configure this in the role trust policy and in the principal's permissions policy.
- When you assume a role with another role, called role chaining, permissions for `sts:SetSourceIdentity` are required in both the permissions policy of the principal who is assuming the role and in the role trust policy of the target role. Otherwise, the assume role operation will fail.

- When using source identity, the role trust policies for all roles connected to an IdP must have the `sts:SetSourceIdentity` permission. The `AssumeRole*` operation will fail for any role connected to an IdP without this permission. If you don't want to update the role trust policy for each role, you can use a separate IdP instance for passing source identity and add the `sts:SetSourceIdentity` permission to only the roles that are connected to the separate IdP.
- To set a source identity across account boundaries, you must include the `sts:SetSourceIdentity` permission in two places. It must be in the permissions policy of the principal in the originating account and in the role trust policy of the role in the target account. You might need to do this, for example, when a role is used to assume a role in another account with role chaining.

As the account administrator, imagine that you want to allow the IAM user `DevUser` in your account to assume the `Developer_Role` in the same account. But you want to allow this action only if the user has set the source identity to their IAM user name. You can attach the following policy to the IAM user.

Example identity-based policy attached to DevUser

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AssumeRole",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::123456789012:role/Developer_Role"
    },
    {
      "Sid": "SetAwsUserNameAsSourceIdentity",
      "Effect": "Allow",
      "Action": "sts:SetSourceIdentity",
      "Resource": "arn:aws:iam::123456789012:role/Developer_Role",
      "Condition": {
        "StringLike": {
          "sts:SourceIdentity": "${aws:username}"
        }
      }
```

```
    }
  ]
}
```

To enforce the acceptable source identity values, you can configure the following role trust policy. The policy gives the IAM user `DevUser` permissions to assume the role and set a source identity. The `sts:SourceIdentity` condition key defines the acceptable source identity value.

Example role trust policy for source identity

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDevUserAssumeRole",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/DevUser"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:SetSourceIdentity"
      ],
      "Condition": {
        "StringEquals": {
          "sts:SourceIdentity": "DevUser"
        }
      }
    }
  ]
}
```

Using the credentials for the IAM user `DevUser`, the user attempts to assume the `DeveloperRole` using the following AWS CLI request.

Example AssumeRole CLI request

```
aws sts assume-role \
```

```
--role-arn arn:aws:iam::123456789012:role/Developer_Role \
--role-session-name Dev-project \
--source-identity DevUser \
```

When AWS evaluates the request, the request context contains the `sts:SourceIdentity` of `DevUser`.

Specifying a source identity when assuming a role

You can specify a source identity when you use one of the AWS STS `AssumeRole*` API operations to get temporary security credentials for a role. The API operation that you use differs depending on your use case.

For example, if you use IAM roles to give IAM users access to AWS resources that they don't normally have access to, you might use the `AssumeRole` operation. If you use enterprise identity federation to manage your workforce users, you might use the `AssumeRoleWithSAML` operation. If you use OIDC federation to allow end users to access your mobile or web applications, you might use the `AssumeRoleWithWebIdentity` operation. The following sections explain how to use source identity with each operation. To learn more about common scenarios for temporary credentials, see Common scenarios for temporary credentials.

Using source identity with AssumeRole

The `AssumeRole` operation returns a set of temporary credentials that you can use to access AWS resources. You can use IAM user or role credentials to call `AssumeRole`. To pass source identity while assuming a role, use the `--source-identity` AWS CLI option or the `SourceIdentity` AWS API parameter. The following example shows how to specify the source identity using the AWS CLI.

Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/developer \
--role-session-name Audit \
--source-identity Admin \
```

Using source identity with AssumeRoleWithSAML

The principal calling the `AssumeRoleWithSAML` operation is authenticated using SAML-based federation. This operation returns a set of temporary credentials that you can use to access AWS

resources. For more information about using SAML-based federation for AWS Management Console access, see Enabling SAML 2.0 federated principals to access the AWS Management Console. For details about AWS CLI or AWS API access, see SAML 2.0 federation. For a tutorial of setting up SAML federation for your Active Directory users, see AWS Federated Authentication with Active Directory Federation Services (ADFS) in the AWS Security Blog.

As an administrator, you can allow members of your company directory to federate into AWS using the AWS STS `AssumeRoleWithSAML` operation. To do this, you must complete the following tasks:

1. Configure a SAML provider in your organization.
2. Create a SAML provider in IAM.
3. Configure a role and its permissions in AWS for your SAML federated principals.
4. Finish configuring the SAML IdP and create assertions for the SAML authentication response.

To set a SAML attribute for source identity, include the `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/SourceIdentity`. Use the `AttributeValue` element to specify the value of the source identity. For example, assume that you want to pass the following identity attribute as the source identity.

```
SourceIdentity:DiegoRamirez
```

To pass this attribute, include the following element in your SAML assertion.

Example snippet of a SAML assertion

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/SourceIdentity">
<AttributeValue>DiegoRamirez</AttributeValue>
</Attribute>
```

Using source identity with AssumeRoleWithWebIdentity

The principal calling the `AssumeRoleWithWebIdentity` operation is authenticated using OpenID Connect (OIDC)-compliant federation. This operation returns a set of temporary credentials that you can use to access AWS resources. For more information about using OIDC federation for AWS Management Console access, see OIDC federation.

To pass source identity from OpenID Connect (OIDC), you must include the source identity in the JSON Web Token (JWT). Include source identity in the `https://aws.amazon.com/` `source_identity` namespace in the token when you submit the

`AssumeRoleWithWebIdentity` request. To learn more about OIDC tokens and claims, see Using Tokens with User Pools in the *Amazon Cognito Developer Guide*.

For example, the following decoded JWT is a token that is used to call `AssumeRoleWithWebIdentity` with the `Admin` source identity.

Example decoded JSON Web Token

```
{
    "sub": "john",
    "aud": "ac_oic_client",
    "jti": "ZYUCeRMQVtqHypVPWAN3VB",
    "iss": "https://xyz.com",
    "iat": 1566583294,
    "exp": 1566583354,
    "auth_time": 1566583292,
    "https://aws.amazon.com/source_identity":"Admin"
}
```

Control access using source identity information

When a source identity is initially set, the sts:SourceIdentity key is present in the request. After a source identity is set, the aws:SourceIdentity key is present in all subsequent requests made during the role session. As the administrator, you can write policies that grant conditional authorization to perform AWS actions based on the existence or value of the source identity attribute.

Imagine that you want to require your developers to set a source identity to assume a critical role that has permission to write to a production critical AWS resource. Also imagine that you grant AWS access to your workforce identities using `AssumeRoleWithSAML`. You only want senior developers Saanvi and Diego to have access to the role, so you create the following trust policy for the role.

Example role trust policy for source identity (SAML)

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "SAMLProviderAssumeRoleWithSAML",
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:saml-provider/name-of-identity-
provider"
      },
      "Action": [
        "sts:AssumeRoleWithSAML"
      ],
      "Condition": {
        "StringEquals": {
          "SAML:aud": "https://region-code.signin.aws.amazon.com/saml"
        }
      }
    },
    {
      "Sid": "SetSourceIdentitySrEngs",
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:saml-provider/name-of-identity-
provider"
      },
      "Action": [
        "sts:SetSourceIdentity"
      ],
      "Condition": {
        "StringLike": {
          "sts:SourceIdentity": [
            "Saanvi",
            "Diego"
          ]
        }
      }
    }
  ]
}
```

The trust policy contains a condition for `sts:SourceIdentity` that requires a source identity of Saanvi or Diego to assume the critical role.

Alternatively, if you use an OIDC provider for federation and users are authenticated with `AssumeRoleWithWebIdentity`, your role trust policy might look as follows.

Example role trust policy for source identity (OIDC provider)

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/server.example.com"
      },
      "Action": [
        "sts:AssumeRoleWithWebIdentity",
        "sts:SetSourceIdentity"
      ],
      "Condition": {
        "StringEquals": {
          "server.example.com:aud": "oidc-audience-id"
        },
        "StringLike": {
          "sts:SourceIdentity": [
            "Saanvi",
            "Diego"
          ]
        }
      }
    }
  ]
}
```

Role chaining and cross-account requirements

Imagine that you want to allow users who have assumed `CriticalRole` to assume a `CriticalRole_2` in another account. The role session credentials that were obtained to assume `CriticalRole` are used to role chain to a second role, `CriticalRole_2`, in a different account. The role is being assumed across an account boundary. Therefore, the `sts:SetSourceIdentity` permission must be granted in both the permissions policy on `CriticalRole` and in the role trust policy on `CriticalRole_2`.

Example permissions policy on CriticalRole

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AssumeRoleAndSetSourceIdentity",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole",
        "sts:SetSourceIdentity"
      ],
      "Resource": "arn:aws:iam::222222222222:role/CriticalRole_2"
    }
  ]
}
```

To secure setting source identity across the account boundary, the following role trust policy trusts only the role principal for `CriticalRole` to set the source identity.

Example role trust policy on CriticalRole_2

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:role/CriticalRole"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:SetSourceIdentity"
      ],
      "Condition": {
        "StringLike": {
```

```
          "aws:SourceIdentity": ["Saanvi","Diego"]
        }
      }
    }
  ]
}
```

The user makes the following call using role session credentials obtained from assuming CriticalRole. The source identity was set during the assumption of CriticalRole, so it does not need to be explicitly set again. If the user attempts to set a source identity that is different from the value set when `CriticalRole` was assumed, the assume role request will be denied.

Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::222222222222:role/CriticalRole_2 \
--role-session-name Audit \
```

When the calling principal assumes the role, the source identity in the request persists from the first assumed role session. Therefore, both the `aws:SourceIdentity` and `sts:SourceIdentity` keys are present in the request context.

Viewing source identity in CloudTrail

You can use CloudTrail to view the requests made to assume roles or federate users. You can also view the role or user requests to take actions in AWS. The CloudTrail log file includes information about the source identity set for the assumed-role or federated user session. For more information, see Logging IAM and AWS STS API calls with AWS CloudTrail

For example, assume that a user makes an AWS STS `AssumeRole` request, and sets a source identity. You can find the `sourceIdentity` information in the `requestParameters` key in your CloudTrail log.

Example requestParameters section in an AWS CloudTrail log

```
"eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSAccount",
        "principalId": "AIDAJ45Q7YFFAREXAMPLE",
        "accountId": "111122223333"
```

```
    },
    "eventTime": "2020-04-02T18:20:53Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRole",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.64",
    "userAgent": "aws-cli/1.16.96 Python/3.6.0 Windows/10 botocore/1.12.86",
    "requestParameters": {
        "roleArn": "arn:aws:iam::123456789012:role/DevRole",
        "roleSessionName": "Dev1",
        "sourceIdentity": "source-identity-value-set"
    }
```

If the user uses the assumed role session to perform an action, the source identity information is present in the `userIdentity` key in the CloudTrail log.

Example userIdentity key in an AWS CloudTrail log

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAJ45Q7YFFAREXAMPLE:Dev1",
    "arn": "arn:aws:sts::123456789012:assumed-role/DevRole/Dev1",
    "accountId": "123456789012",
    "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAJ45Q7YFFAREXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/DevRole",
        "accountId": "123456789012",
        "userName": "DevRole"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2021-02-21T23:46:28Z"
      },
      "sourceIdentity": "source-identity-value-present"
    }
  }
}
```

To see example AWS STS API events in CloudTrail logs, see Example IAM API events in CloudTrail log. For more details about the information contained in CloudTrail log files, see CloudTrail Event Reference in the *AWS CloudTrail User Guide*.

Permissions for GetFederationToken

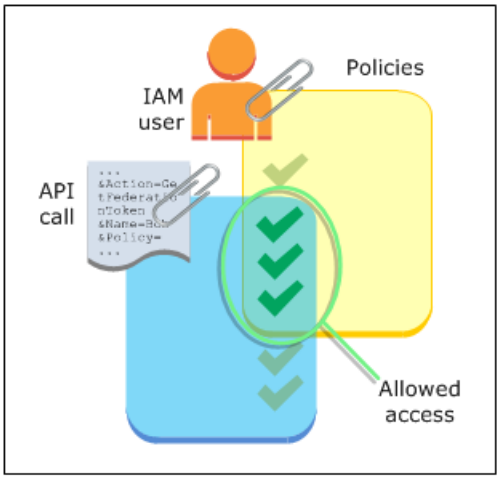
The `GetFederationToken` operation is called by an IAM user and returns temporary credentials for that user. This operation *federates* the user. The permissions assigned to an AWS STS federated user session are defined in one of two places:

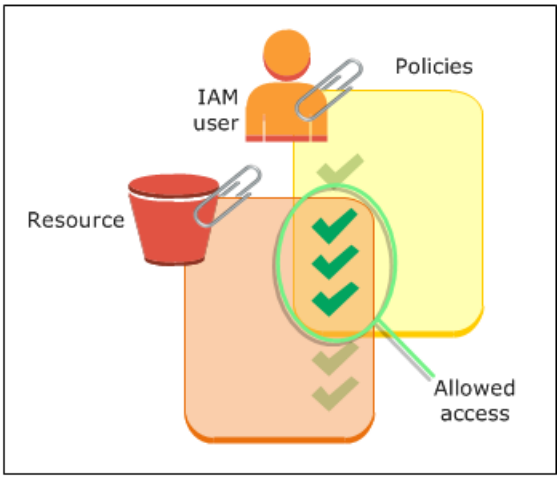
- The session policies passed as a parameter of the `GetFederationToken` API call. (This is most common.)
- A resource-based policy that explicitly names the AWS STS federated user session in the `Principal` element of the policy. (This is less common.)

Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session. When you create an AWS STS federated user session and pass session policies, the resulting session's permissions are the intersection of the user's identity-based policy and the session policies. You cannot use the session policy to grant more permissions than those allowed by the identity-based policy of the user that is being federated.

In most cases if you do not pass a policy with the `GetFederationToken` API call, the resulting temporary security credentials have no permissions. However, a resource-based policy can provide additional permissions for the session. You can access a resource with a resource-based policy that specifies your session as the allowed principal.

The following figures show a visual representation of how the policies interact to determine permissions for the temporary security credentials returned by a call to `GetFederationToken`.





Example: Assigning permissions using GetFederationToken

You can use the `GetFederationToken` API action with different kinds of policies. Here are a few examples.

Policy attached to the IAM user

In this example, you have a browser-based client application that relies on two backend web services. One backend service is your own authentication server that uses your own identity system to authenticate the client application. The other backend service is an AWS service that provides some of the client application's functionality. The client application is authenticated by your server, and your server creates or retrieves the appropriate permissions policy. Your server then calls the `GetFederationToken` API to obtain temporary security credentials, and returns those credentials to the client application. The client application can then make requests directly to the AWS service with the temporary security credentials. This architecture allows the client application to make AWS requests without embedding long-term AWS credentials.

Your authentication server calls the `GetFederationToken` API with the long-term security credentials of an IAM user named `token-app`. But the long-term IAM user credentials remain on your server and are never distributed to the client. The following example policy is attached to the `token-app` IAM user and defines the broadest set of permissions that your AWS STS federated users (clients) will need. Note that the `sts:GetFederationToken` permission is required for your authentication service to obtain temporary security credentials for the AWS STS federated users.

Example policy attached to IAM user token-app that calls GetFederationToken

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:GetFederationToken",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "dynamodb:ListTables",
      "Resource": "*"
    },
```

```
    {
      "Effect": "Allow",
      "Action": "sqs:ReceiveMessage",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "sns:ListSubscriptions",
      "Resource": "*"
    }
  ]
}
```

The preceding policy grants several permissions to the IAM user. However, this policy alone doesn't grant any permissions to the AWS STS federated user. If this IAM user calls `GetFederationToken` and does not pass a policy as a parameter of the API call, the resulting AWS STS federated user has no effective permissions.

Session policy passed as parameter

The most common way to ensure that the AWS STS federated user is assigned appropriate permission is to pass session policies in the `GetFederationToken` API call. Expanding on the previous example, imagine that `GetFederationToken` is called with the credentials of the IAM user `token-app`. Then imagine that the following session policy is passed as a parameter of the API call. The resulting AWS STS federated user has permission to list the contents of the Amazon S3 bucket named `productionapp`. The user can't perform the Amazon S3 `GetObject`, `PutObject`, and `DeleteObject` actions on items in the `productionapp` bucket.

The federated user is assigned these permissions because the permissions are the intersection of the IAM user policies and the session policies that you pass.

The AWS STS federated user could not perform actions in Amazon SNS, Amazon SQS, Amazon DynamoDB, or in any S3 bucket except `productionapp`. These actions are denied even though those permissions are granted to the IAM user that is associated with the `GetFederationToken` call.

Example session policy passed as parameter of GetFederationToken API call

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::productionapp"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": ["arn:aws:s3:::productionapp/*"]
    }
  ]
}
```

Resource-based policies

Some AWS resources support resource-based policies, and these policies provide another mechanism to grant permissions directly to an AWS STS federated user. Only some AWS services support resource-based policies. For example, Amazon S3 has buckets, Amazon SNS has topics, and Amazon SQS has queues that you can attach policies to. For a list of all services that support resource-based policies, see AWS services that work with IAM and review the "Resource-based policies" column of the tables. You can use resource-based policies to assign permissions directly to an AWS STS federated user. Do this by specifying the Amazon Resource Name (ARN) of the AWS STS federated user in the `Principal` element of the resource-based policy. The following example illustrates this and expands on the previous examples, using an S3 bucket named `productionapp`.

The following resource-based policy is attached to the bucket. This bucket policy allows an AWS STS federated user named Carol to access the bucket. When the example policy described earlier is attached to the `token-app` IAM user, the AWS STS federated user named Carol has permission

to perform the `s3:GetObject`, `s3:PutObject`, and `s3:DeleteObject` actions on the bucket named `productionapp`. This is true even when no session policy is passed as a parameter of the `GetFederationToken` API call. That's because in this case the AWS STS federated user named Carol has been explicitly granted permissions by the following resource-based policy.

Remember, an AWS STS federated user is granted permissions only when those permissions are explicitly granted to both the IAM user ***and*** the AWS STS federated user. They can also be granted (within the account) by a resource-based policy that explicitly names the AWS STS federated user in the `Principal` element of the policy, as in the following example.

Example bucket policy that allows access to federated user

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Principal": {
            "AWS": "arn:aws:sts::111122223333:federated-user/Carol"
        },
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:PutObject",
            "s3:DeleteObject"
        ],
        "Resource": [
            "arn:aws:s3:::productionapp/*"
        ]
    }
}
```

For more information about how policies are evaluated see Policy evaluation logic.

Permissions for GetSessionToken

The primary occasion for calling the `GetSessionToken` API operation or the `get-session-token` CLI command is when a user must be authenticated with multi-factor authentication (MFA). It is possible to write a policy that allows certain actions only when those actions are requested by a user who has been authenticated with MFA. In order to successfully pass the

MFA authorization check, a user must first call `GetSessionToken` and include the optional `SerialNumber` and `TokenCode` parameters. If the user is successfully authenticated with an MFA device, the credentials returned by the `GetSessionToken` API operation include the MFA context. This context indicates that the user is authenticated with MFA and is authorized for API operations that require MFA authentication.

Permissions required for GetSessionToken

No permissions are required for a user to get a session token. The purpose of the `GetSessionToken` operation is to authenticate the user using MFA. You cannot use policies to control authentication operations.

To grant permissions to perform most AWS operations, you add the action with the same name to a policy. For example, to create a user, you must use the `CreateUser` API operation, the `create-user` CLI command, or the AWS Management Console. To perform these operations, you must have a policy that allows you to access the `CreateUser` action.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateUser",
            "Resource": "*"
        }
    ]
}
```

You can include the `GetSessionToken` action in your policies, but it has no effect on a user's ability to perform the `GetSessionToken` operation.

Permissions granted by GetSessionToken

If `GetSessionToken` is called with the credentials of an IAM user, the temporary security credentials have the same permissions as the IAM user. Similarly, if `GetSessionToken` is called with AWS account root user credentials, the temporary security credentials have root user permissions.

> **Note**
>
> We recommend that you do not call `GetSessionToken` with root user credentials. Instead, follow our best practices and create IAM users with the permissions they need. Then use these IAM users for everyday interaction with AWS.

The temporary credentials that you get when you call `GetSessionToken` have the following capabilities and limitations:

- You can use the credentials to access the AWS Management Console by passing the credentials to the federation single sign-on endpoint at `https://signin.aws.amazon.com/federation`. For more information, see Enable custom identity broker access to the AWS console.
- You **cannot** use the credentials to call IAM or AWS STS API operations. You **can** use them to call API operations for other AWS services.

Compare this API operation and its limitations and capability with the other API operations that create temporary security credentials at Compare AWS STS credentials

For more information about MFA-protected API access using `GetSessionToken`, see Secure API access with MFA.

Disabling permissions for temporary security credentials

Temporary security credentials are valid until they expire. These credentials are valid for the specified duration, from 900 seconds (15 minutes) up to a maximum of 129,600 seconds (36 hours). The default session duration is 43,200 seconds (12 hours). You can revoke these credentials, but you must also change permissions for the IAM user or role to stop the use of compromised credentials for malicious account activity. Permissions assigned to temporary security credentials are evaluated each time they are used to make an AWS request. Once you remove all permissions from the credentials, AWS requests that use them fail.

It might take a few minutes for policy updates to take effect. For IAM role sessions, you can revoke the role's temporary security credentials to force all users assuming the role to reauthenticate and request new credentials. For more information, see Revoke the role's temporary security credentials.

You cannot change the permissions for an AWS account root user. Likewise, you cannot change the permissions for the temporary security credentials that were created by calling `GetFederationToken` or `GetSessionToken` while signed in as the root user. For this reason, we recommend that you do not call `GetFederationToken` or `GetSessionToken` as a root user.

For procedures on how to change the permissions for an IAM user, see Change permissions for an IAM user.

For procedures on how to change the permissions for an IAM role, see Update permissions for a role.

> **Important**
>
> You can't edit roles in IAM that were created from IAM Identity Center permission sets. You must revoke the active permission set session for a user in IAM Identity Center. For more information, see Revoke active IAM role sessions created by permission sets in the *IAM Identity Center User Guide*.

Topics

- Deny access to all IAM role sessions associated with a role
- Deny access to a specific IAM role session
- Deny access to temporary security credentials sessions with condition context keys
- Deny access to a specific principal with resource-based policies

Deny access to all IAM role sessions associated with a role

This procedure denies permissions to **all** IAM role sessions associated with a role. Use this approach when you are concerned about suspicious access by:

- Principals from another account using cross-account access
- External user identities with permissions to access AWS resources in your account
- Users who have been authenticated in a mobile or web application with an OIDC provider

To change or remove the permissions assigned to the temporary security credentials obtained by calling `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity`,

`GetFederationToken`, or `GetSessionToken`, you can edit or delete the identity-based policy that defines the permissions for the role.

> **Important**
>
> If there's a resource-based policy that allows the principal access, you must also add an explicit deny for that resource. See Deny access to a specific principal with resource-based policies for details.

To deny access to all IAM role sessions associated with a role

1. Sign in to the AWS Management Console and open the IAM console.
2. In the navigation pane, choose **Roles**..
3. Choose the name of the role to edit. You can use the search box to filter the list.
4. Choose the **Permissions** tab.
5. Select the relevant policy to edit. Before you edit a customer managed policy, review the **Entities attached** tab to avoid disrupting access to other identities that may have the same policy attached.
6. Choose the **JSON** tab and update the policy to deny all resources and actions.

> **Note**
>
> These permissions are the same as those in the AWS managed policy AWSDenyAll. You can attach this AWS managed policy to any IAM user or role you want to deny all access to.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DenyAll",
            "Effect": "Deny",
            "Action": [
```

```
                "*"
            ],
            "Resource": "*"
        }
    ]
}
```

7. On the **Review** page, review the policy **Summary** and then choose **Save changes** to save your work.

When you update the policy, the changes affect the permissions of all temporary security credentials associated with the role, including credentials that were issued before you changed the role's permissions policy.

After you update the policy, you can revoke the role's temporary security credentials to immediately revoke all permissions to the role's issued credentials.

Deny access to a specific IAM role session

When you update IAM roles with a deny-all policy or delete the role entirely, all users that have access to the role are disrupted. You can deny access without impacting the permissions of all other sessions associated with the role.

The `Principal` can be denied permissions using condition context keys or resource-based policies.

> **Tip**
>
> You can find the ARNs of federated users using AWS CloudTrail logs. For more information, see How to Easily Identify Your Federated Users by Using AWS CloudTrail.

Deny access to temporary security credentials sessions with condition context keys

You can use condition context keys in identity-based policies in situations where you want to deny access to specific temporary security credential sessions without affecting the permissions of the IAM user or role that created the credentials. For IAM roles, after you update the policy, you can also revoke the role's temporary security credentials sessions to immediately revoke all issued credentials.

For more information about condition context keys, see AWS global condition context keys.

aws:PrincipalArn

You can use condition context key aws:PrincipalArn in an identity-based policy to deny access to a specific principal by their Amazon Resource Name (ARN). You do this by specifying the ARN of the IAM user, role, or AWS STS federated user session the temporary security credentials are associated with in the Condition element of a policy.

To deny access to a specific principal by their ARN

1. In the IAM console navigation pane, choose **Users** or **Roles**.
2. Choose the name of the IAM user or role to edit. You can use the search box to filter the list.
3. Choose the **Permissions** tab.
4. Select the relevant policy to edit. Before you edit a customer managed policy, review the **Entities attached** tab to avoid disrupting access to other identities that may have the same policy attached.
5. Choose the **JSON** tab and add a deny statement for the principal ARN as shown in the following example.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "aws:PrincipalArn": [
            "arn:aws:iam::222222222222:role/ROLENAME",
            "arn:aws:iam::222222222222:user/USERNAME",
            "arn:aws:iam::222222222222:federated-user/USERNAME"
          ]
        }
      }
    }
  ]
}
```

6. On the **Review** page, review the policy **Summary** and then choose **Save changes** to save your work.

aws:SourceIdentity

You can use condition context key aws:SourceIdentity in an identity-based policy to deny access to a specific source identity associated with an IAM role session. This applies as long as the role session was issued by setting the `SourceIdentity` request parameter when the principal assumed a role using any AWS STS `assume-role*` CLI commands, or AWS STS `AssumeRole*` API operations. You do this by specifying the source identity that the temporary security credentials are associated with in the `Condition` element of a policy.

Unlike context key sts:RoleSessionName, after the source identity is set, the value cannot be changed. The `aws:SourceIdentity` key is present in the request context for all actions taken by the role. The source identity persists into subsequent role sessions when you use the session credentials to assume another role. Assuming one role from another is called role chaining.

The following policy shows an example of how you can deny access to temporary security credential sessions using condition context key `aws:SourceIdentity`. If you specify the source identity associated with a role session, it will deny role sessions with the named source identity without affecting the permissions of the role that created the credentials. For this example, the source identity set by the principal when the role session was issued is `nikki_wolf@example.com`. Any request made by a role session with the source identity `nikki_wolf@example.com` will be denied because the source identity is included in the policy condition and the policy Effect is set to `Deny`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:SourceIdentity": [
            "nikki_wolf@example.com",
```

```
            "<source identity value>"
          ]
        }
      }
    }
  ]
}
```

aws:userid

You can use condition context key aws:userid in an identity-based policy to deny access to all or specific temporary security credential sessions associated with the IAM user or role. You do this by specifying the unique identifier (ID) of the IAM user, role, or AWS STS federated user session the temporary security credentials are associated with in the `Condition` element of a policy.

The following policy shows an example of how you can deny access to temporary security credential sessions using condition context key `aws:userid`.

- `AIDAXUSER1` represents the unique ID for an IAM user. Specifying the unique ID of an IAM user as a value for context key `aws:userid` will deny access to the IAM user. This includes any temporary security credential sessions that were created by calling the `GetSessionToken` API.
- `AROAXROLE1:*` represents the unique ID for all sessions associated with the IAM role. Specifying the unique ID of an IAM role and a wildcard (*) character in the caller-specified-role-session-name portion as a value for context key `aws:userid` will deny all sessions associated with the role.
- `AROAXROLE2:<caller-specified-role-session-name>` represents the unique ID for an assumed-role session. In the caller-specified-role-session-name portion of the assumed-role unique ID you can specify a role session name or a wildcard character if the StringLike condition operator is used. If you specify the role session name, it will deny the named role session without affecting the permissions of the role that created the credentials. If you specify a wildcard character for the role session name, it will deny all sessions associated with the role.

> **Note**
>
> The caller-specified role session name, which is part of the unique identifier for an assumed-role session, can change during role chaining. Role chaining occurs when one role assumes another role. The role session name is set using the `RoleSessionName`

request parameter when the principal assumes a role using the AWS STS `AssumeRole` API operation.

- `account-id:<federated-user-caller-specified-name>` represents the unique ID for an AWS STS federated user session. An IAM user creates this session by calling the `GetFederationToken` API. Specifying the unique ID for an AWS STS federated user session denies the named federated session without affecting the permissions of the IAM user that created the credentials.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:userId": [
            "AIDAXUSER1",
            "AROAXROLE1:*",
            "AROAXROLE2:<caller-specified-role-session-name>",
            "123456789012:<federated-user-caller-specified-name>"
          ]
        }
      }
    }
  ]
}
```

For specific examples of principal key values, see Principal key values. For information about IAM unique identifiers and how to get them, see Unique identifiers.

Deny access to a specific principal with resource-based policies

To restrict access to a specific principal with a resource-based policy, you can use condition context keys aws:PrincipalArn or aws:SourceIdentity in the `Condition` element. A resource-based policy

is a permission policy attached to a resource and controls who can access the resource and what actions they can perform on it.

When you use the `aws:PrincipalARN` context key, specify the ARN of the IAM user, role, or AWS STS federated user session associated with the temporary security credentials in the Condition element of a policy. The following example policy demonstrates how to use the `aws:PrincipalARN` context key in a resource-based policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Principal": "*",
    "Effect": "Deny",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
    "Condition": {
      "ArnEquals": {
        "aws:PrincipalArn": [
          "arn:aws:iam::222222222222:role/ROLENAME",
          "arn:aws:iam::222222222222:user/USERNAME",
          "arn:aws:sts::222222222222:federated-user/USERNAME"
        ]
      }
    }
  }
}
```

When you use the `aws:SourceIdentity` context key, specify the source identity value associated with the role's temporary security credentials in the `Condition` element of a policy. This applies as long as the role session was issued by setting the `SourceIdentity` request parameter when the principal assumed a role using any AWS STS `assume-role*` CLI commands, or AWS STS `AssumeRole*` API operations. The following example demonstrates how to use the `aws:SourceIdentity` context key in a resource-based policy:

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": {
    "Principal": "*",
    "Effect": "Deny",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
    "Condition": {
      "StringLike": {
        "aws:SourceIdentity": [
          "nikki_wolf@example.com",
          "<source identity value>"
        ]
      }
    }
  }
}
```

If you only update the identity-based policy for a principal, they can still perform actions allowed in the resource-based policy, except when those actions are explicitly denied in the identity-based policy.

To deny access to a specific principal in a resource-based policy

1. Refer to AWS services that work with IAM to see if the service supports resource-based policies.
2. Sign in to the AWS Management Console and open the console for the service. Each service has a different location in the console for attaching policies.
3. Edit the resource-based policy. Add a deny policy statement to specify the identifying information of the credential:

   a. In the `Principal` element, enter wildcard (*). The principal will be restricted in the `Condition` element.

   b. In the `Effect` element, enter “Deny.”

   c. In `Action`, enter the service namespace and the name of the action to deny. To deny all actions, use the wildcard (*) character. For example: `"s3:*"`.

   d. In the `Resource` element, enter the ARN of the target resource. For example: `"arn:aws:s3:::amzn-s3-demo-bucket"`.

e. In the `Condition` element, specify either the `aws:PrincipalARN` or `aws:SourceIdentity` context key.

   If you use the `aws:PrincipalARN` context key, enter the ARN of the principal to deny access for.

   If you use the `aws:SourceIdentity` context key, enter the source identity value set in the role session to deny access for.

4. Save your work.

Granting permissions to create temporary security credentials

By default, IAM users do not have permission to create temporary security credentials for AWS STS federated user sessions and roles. You must use a policy to provide your users with these permissions. Although you can grant permissions directly to a user, we strongly recommend that you grant permissions to a group. This makes management of the permissions much easier. When someone no longer needs to perform the tasks associated with the permissions, you simply remove them from the group. If someone else needs to perform that task, add them to the group to grant the permissions.

To grant an IAM group permission to create temporary security credentials for AWS STS federated user sessions or roles, you attach a policy that grants one or both of the following privileges:

- For OIDC and SAML federated principals to access an IAM role, grant access to AWS STS `AssumeRole`.
- For AWS STS federated users that don't need a role, grant access to AWS STS `GetFederationToken`.

For more information about the differences between the `AssumeRole` and `GetFederationToken` API operations, see Request temporary security credentials.

IAM users can also call `GetSessionToken` to create temporary security credentials. No permissions are required for a user to call `GetSessionToken`. The purpose of this operation is to authenticate the user using MFA. You cannot use policies to control authentication. This means that you cannot prevent IAM users from calling `GetSessionToken` to create temporary credentials.

Example policy that grants permission to assume a role

The following example policy grants permission to call `AssumeRole` for the `UpdateApp` role in AWS account `123123123123`. When `AssumeRole` is used, the user (or application) that creates the security credentials on behalf of a federated user cannot delegate any permissions that are not already specified in the role permission policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::123123123123:role/UpdateAPP"
  }]
}
```

Example policy that grants permission to create temporary security credentials for a federated user

The following example policy grants permission to access `GetFederationToken`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sts:GetFederationToken",
    "Resource": "*"
  }]
}
```

Important

When you give IAM users permission to create temporary security credentials for AWS STS federated users with `GetFederationToken`, be aware that this permits those users

to delegate their own permissions. For more information about delegating permissions across IAM users and AWS accounts, see Examples of policies for delegating access. For more information about controlling permissions in temporary security credentials, see Permissions for temporary security credentials.

Example policy that grants a user limited permission to create temporary security credentials for federated users

When you let an IAM user call `GetFederationToken`, it is a best practice to restrict the permissions that the IAM user can delegate. For example, the following policy shows how to let an IAM user create temporary security credentials only for AWS STS federated users whose names start with *Manager*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sts:GetFederationToken",
    "Resource": ["arn:aws:sts::123456789012:federated-user/Manager*"]
  }]
}
```

Granting permissions to use identity-enhanced console sessions

Identity-enhanced console sessions enables AWS IAM Identity Center user and session IDs to be included in users' AWS console sessions when they sign in. For example, Amazon Q Developer Pro uses identity-enhanced console sessions to personalize the service experience. For more information about identity-enhanced console sessions, see Enabling identity-enhanced console sessions in the *AWS IAM Identity Center User Guide*. For information about Amazon Q Developer setup, see Setting up Amazon Q Developer in the *Amazon Q Developer User Guide*.

For identity-enhanced console sessions to be available to a user, you must use an identity-based policy to grant the IAM principal the `sts:SetContext` permission for the resource that represents their own console session.

Important

By default, users do not have permission to set context for their identity-enhanced console sessions. To allow this, you must grant the IAM principal the `sts:SetContext` permission in an identity-based policy as shown in the policy example below.

The following example identity-based policy grants the `sts:SetContext` permission to an IAM principal, allowing the principal to set identity-enhanced console session context for their own AWS console sessions. The policy resource, `arn:aws:sts::account-id:self`, represents the caller's AWS session. The `account-id` ARN segment can be replaced with a wildcard character * in cases where the same permission policy is deployed across multiple accounts, such as when this policy is deployed using IAM Identity Center permission sets.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:SetContext",
            "Resource": "arn:aws:sts::111122223333:self"
        }
    ]
}
```

Manage AWS STS in an AWS Region

A Regional endpoint is the URL of the entry point within a particular region for an AWS web service. AWS recommends using Regional AWS Security Token Service (AWS STS) endpoints instead of the global endpoint to reduce latency, build in redundancy, and increase session token validity.

Although the global (legacy) AWS STS endpoint `https://sts.amazonaws.com` is highly available, it's hosted in a single AWS Region, US East (N. Virginia), and like other endpoints, it doesn't provide automatic failover to endpoints in other Regions.

- **Reduce latency** – By making your AWS STS calls to an endpoint that is geographically closer to your services and applications, you can access AWS STS services with lower latency and better response times.
- **Build in redundancy** – You can limit the effects of a failure within a workload to a limited number of components with a predictable scope of impact containment. Using regional AWS STS endpoints lets you align the scope of your components with the scope of your session tokens. For more information about this reliability pillar, see Use fault isolation to protect your workload in the *AWS Well-Architected Framework*.
- **Increase session token validity** – Session tokens from Regional AWS STS endpoints are valid in all AWS Regions. Session tokens from the global STS endpoint are valid only in AWS Regions that are enabled by default. If you intend to enable a new Region for your account, you can use session tokens from Regional AWS STS endpoints. If you choose to use the global endpoint, you must change the Region compatibility of AWS STS session tokens for the global endpoint. Doing so ensures that tokens are valid in all AWS Regions.

For a list of AWS STS Regions and their endpoints, see AWS STS Regions and endpoints.

> **Note**
>
> AWS has made changes to the AWS Security Token Service (AWS STS) global endpoint (`https://sts.amazonaws.com`) in Regions enabled by default to enhance its resiliency and performance. AWS STS requests to the global endpoint are automatically served in the same AWS Region as your workloads. These changes will not be deployed to opt-in Regions. We recommend that you use the appropriate AWS STS regional endpoints. For more information, see AWS STS global endpoint changes.

Topics

- Activating and deactivating AWS STS in an AWS Region
- Writing code to use AWS STS Regions
- Managing global endpoint session tokens

Activating and deactivating AWS STS in an AWS Region

When you activate AWS STS endpoints for a Region, AWS STS can issue temporary credentials to users and roles in your account that make an AWS STS request. Those credentials can then be used in any Region that is enabled by default or is manually enabled.

For Regions that are enabled by default, you must activate the Regional AWS STS endpoint in the account where the temporary credentials are generated. It does not matter whether a user is signed into the same account or a different account when they make the request.

When requesting temporary credentials for a role in another AWS account using a Region that is manually enabled, the target account (the account containing the role) must enable that Region for AWS STS operations. This ensures that the temporary security credentials can be generated correctly.

For example, imagine a user in account A wants to send an `sts:AssumeRole` API request to the AWS STS Regional endpoint `https://sts.ap-southeast-3.amazonaws.com`. The request is for temporary credentials for the role named `Developer` in account B. Because the request is to create credentials for an entity in account B, account B must have the `ap-southeast-3` Region enabled. Users from account A (or any other account) can call the `ap-southeast-3` AWS STS endpoint to request credentials for account B whether or not the Region is activated in their accounts. To learn more, see Enable or disable AWS Regions in your account.

> **Note**
>
> Active Regions are available to everyone that uses temporary credentials in that account. To control which IAM users or roles can access the Region, use the `aws:RequestedRegion` condition key in your permissions policies.

To activate or deactivate AWS STS in a Region that is enabled by default (console)

1. Sign in as a root user or a user with permissions to perform IAM administration tasks.
2. Open the IAM console and in the navigation pane choose **Account settings**.
3. In the **Security Token Service (STS)** section **Endpoints**, find the Region that you want to configure, and then choose **Active** or **Inactive** in the **STS status** column.
4. In the dialog box that opens, choose **Activate** or **Deactivate**.

For Regions that must be enabled, we activate AWS STS automatically when you enable the Region. After you enable a Region, AWS STS is always active for the Region and you cannot deactivate it. To learn about enabling Regions that are disabled by default, see Specifying which AWS Regions your account can use in the *AWS Account Management Reference Guide*.

Writing code to use AWS STS Regions

After you activate a Region, you can direct AWS STS API calls to that Region. The following Java code snippet demonstrates how to configure an `AWSSecurityTokenService` object to make requests to the Europe (Milan) (eu-south-1) Region.

```
EndpointConfiguration regionEndpointConfig = new EndpointConfiguration("https://sts.eu-
south-1.amazonaws.com", "eu-south-1");
AWSSecurityTokenService stsRegionalClient =
 AWSSecurityTokenServiceClientBuilder.standard()
.withCredentials(credentials)
.withEndpointConfiguration(regionEndpointConfig)
.build();
```

AWS STS recommends that you make calls to a Regional endpoint. To learn how to manually enable a Region, see Specify which AWS Regions your account can use in the *AWS Account Management Reference Guide*.

In the example, the first line instantiates an `EndpointConfiguration` object called `regionEndpointConfig`, passing the URL of the endpoint and the AWS Region as the parameters.

To learn how to set AWS STS regional endpoints using an environment variable for AWS SDKs, see AWS STS Regionalized endpoints in the *AWS SDKs and Tools Reference Guide*.

For all other language and programming environment combinations, refer to the documentation for the relevant SDK.

Managing global endpoint session tokens

Most AWS Regions are enabled for operations in all AWS services by default. Those Regions are automatically activated for use with AWS STS. Some Regions, such as Asia Pacific (Hong Kong), must be manually enabled. To learn more about enabling and disabling AWS Regions, see Specify which AWS Regions your account can use in the *AWS Account Management Reference Guide*.

When you enable these AWS Regions, they are automatically activated for use with AWS STS. You cannot activate the AWS STS endpoint for a Region that is disabled. Session tokens that are valid in all AWS Regions include more characters than tokens that are valid in Regions that are enabled by default. Changing this setting might affect existing systems where you temporarily store tokens.

You can change this setting using the AWS Management Console, AWS CLI, or AWS API.

To change the Region compatibility of session tokens for the global endpoint (console)

1. Sign in as a root user or a user with permissions to perform IAM administration tasks. To change the compatibility of session tokens, you must have a policy that allows the `iam:SetSecurityTokenServicePreferences` action.
2. Open the IAM console. In the navigation pane, choose **Account settings**.
3. Under **Security Token Service (STS)** section **Session Tokens from the STS endpoints**. The **Global endpoint** indicates `Valid only in AWS Regions enabled by default`. Choose **Change**.
4. In the **Change region compatibility** dialog box, select **All AWS Regions**. Then choose **Save changes**.

> **Note**
>
> Session tokens that are valid in all AWS Region include more characters than tokens that are valid in Regions that are enabled by default. Changing this setting might affect existing systems where you temporarily store tokens.

To change the Region compatibility of session tokens for the global endpoint (AWS CLI)

Set the session token version. Version 1 tokens are valid only in AWS Regions that are available by default. These tokens do not work in manually enabled Regions, such as Asia Pacific (Hong Kong). Version 2 tokens are valid in all Regions. However, version 2 tokens include more characters and might affect systems where you temporarily store tokens.

- `aws iam set-security-token-service-preferences`

To change the Region compatibility of session tokens for the global endpoint (AWS API)

Set the session token version. Version 1 tokens are valid only in AWS Regions that are available by default. These tokens do not work in manually enabled Regions, such as Asia Pacific (Hong Kong). Version 2 tokens are valid in all Regions. However, version 2 tokens include more characters and might affect systems where you temporarily store tokens.

- `SetSecurityTokenServicePreferences`

AWS STS Regions and endpoints

> **Note**
>
> AWS has made changes to the AWS Security Token Service (AWS STS) global endpoint (`https://sts.amazonaws.com`) in Regions enabled by default to enhance its resiliency and performance. AWS STS requests to the global endpoint are automatically served in the same AWS Region as your workloads. These changes will not be deployed to opt-in Regions. We recommend that you use the appropriate AWS STS regional endpoints. For more information, see AWS STS global endpoint changes.

The following table lists the Regions and their endpoints. It indicates which ones are activated by default and which ones you can activate or deactivate.

Region name	Endpoint	Active by default	Manually activate/ deactivate
--Global--	sts.amazonaws.com	✓ Yes	✗ No
US East (Ohio)	sts.us-east-2.amazonaws.com	✓ Yes	✓ Yes
US East (N. Virginia)	sts.us-east-1.amazonaws.com	✓ Yes	✗ No
US West (N. California)	sts.us-west-1.amazonaws.com	✓ Yes	✓ Yes
US West (Oregon)	sts.us-west-2.amazonaws.com	✓ Yes	✓ Yes
Africa (Cape Town)	sts.af-south-1.amazonaws.com	✗ No[1]	✗ No

Region name	Endpoint	Active by default	Manually activate/ deactivate
Asia Pacific (Hong Kong)	sts.ap-east-1.amazonaws.com	⊗ No[1]	⊗ No
Asia Pacific (Hyderabad)	sts.ap-south-2.amazonaws.com	⊗ No[1]	⊗ No
Asia Pacific (Jakarta)	sts.ap-southeast-3.amazonaws.com	⊗ No[1]	⊗ No
Asia Pacific (Malaysia)	sts.ap-southeast-5.amazonaws.com	⊗ No[1]	⊗ No
Asia Pacific (Melbourne)	sts.ap-southeast-4.amazonaws.com	⊗ No[1]	⊗ No
Asia Pacific (Mumbai)	sts.ap-south-1.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Osaka)	sts.ap-northeast-3.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Seoul)	sts.ap-northeast-2.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Singapore)	sts.ap-southeast-1.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Sydney)	sts.ap-southeast-2.amazonaws.com	✓ Yes	✓ Yes
Asia Pacific (Thailand)	sts.ap-southeast-7.amazonaws.com	⊗ No[1]	⊗ No
Asia Pacific (Tokyo)	sts.ap-northeast-1.amazonaws.com	✓ Yes	✓ Yes
Canada (Central)	sts.ca-central-1.amazonaws.com	✓ Yes	✓ Yes

Region name	Endpoint	Active by default	Manually activate/ deactivate
Canada West (Calgary)	sts.ca-west-1.amazonaws.com	⊗ No[1]	⊗ No
China (Beijing)	sts.cn-north-1.amazonaws.com.cn	⊙ Yes[2]	⊗ No
China (Ningxia)	sts.cn-northwest-1.amazonaws.com.cn	⊙ Yes[2]	⊙ Yes
Europe (Frankfurt)	sts.eu-central-1.amazonaws.com	⊙ Yes	⊙ Yes
Europe (Ireland)	sts.eu-west-1.amazonaws.com	⊙ Yes	⊙ Yes
Europe (London)	sts.eu-west-2.amazonaws.com	⊙ Yes	⊙ Yes
Europe (Milan)	sts.eu-south-1.amazonaws.com	⊗ No[1]	⊗ No
Europe (Paris)	sts.eu-west-3.amazonaws.com	⊙ Yes	⊙ Yes
Europe (Spain)	sts.eu-south-2.amazonaws.com	⊗ No[1]	⊗ No
Europe (Stockholm)	sts.eu-north-1.amazonaws.com	⊙ Yes	⊙ Yes
Europe (Zurich)	sts.eu-central-2.amazonaws.com	⊗ No[1]	⊗ No
Israel (Tel Aviv)	sts.il-central-1.amazonaws.com	⊗ No[1]	⊗ No
Mexico (Central)	sts.mx-central-1.amazonaws.com	⊗ No[1]	⊗ No
Middle East (Bahrain)	sts.me-south-1.amazonaws.com	⊗ No[1]	⊗ No
Middle East (UAE)	sts.me-central-1.amazonaws.com	⊗ No[1]	⊗ No
South America (São Paulo)	sts.sa-east-1.amazonaws.com	⊙ Yes	⊙ Yes

[1]You must enable the Region to use it. This automatically activates AWS STS. You cannot manually activate or deactivate AWS STS in these Regions.

[2]To use AWS in China, you need an account and credentials specific to AWS in China.

AWS STS global endpoint changes

AWS has made changes to the AWS Security Token Service (AWS STS) global endpoint (`https://sts.amazonaws.com`) in Regions enabled by default to enhance its resiliency and performance. Previously, all requests to the AWS STS global endpoint were served by a single AWS Region, US East (N. Virginia). Now in Regions enabled by default, requests to the AWS STS global endpoint are automatically served in the same Region where the request originates, rather than the US East (N. Virginia) Region. These changes will not be deployed to opt-in Regions.

With this change, AWS STS will process your request based on the originating Region and DNS resolver used. Requests to the AWS STS global endpoint are served in the same Region as your AWS deployed workload if the DNS request for the AWS STS global endpoint is handled by the Amazon DNS server in Regions that are enabled by default. Requests to the AWS STS global endpoint will continue to be served in US East (N. Virginia) Region if your request originated from opt-in Regions or if your request was resolved using a DNS resolver other than the Amazon DNS server. For more information about Amazon DNS, see Amazon DNS server in the *Amazon Virtual Private Cloud User Guide*.

The following table shows how requests to the AWS STS global endpoint are routed based on your DNS provider.

DNS Resolver	Requests to the AWS STS global endpoint routed to the local AWS Region?
Amazon DNS resolver in a Amazon VPC in an Region enabled by default	Yes
Amazon DNS resolver in a Amazon VPC in an opt-in Region	No, the request will be routed to the US East (N. Virginia) Region
DNS resolver provided by your ISP, a public DNS provider, or any other DNS provider	No, the request will be routed to the US East (N. Virginia) Region

To ensure minimal disruption to your existing processes, AWS has implemented the following measures:

- AWS CloudTrail logs for requests made to the AWS STS global endpoint are sent to the US East (N. Virginia) Region. CloudTrail logs for requests served by AWS STS Regional endpoints will continue to be logged to their respective Region in CloudTrail.
- CloudTrail logs for operations performed by the AWS STS global endpoint and Regional endpoints have additional fields `endpointType` and `awsServingRegion` to indicate which endpoint and Region served the request. For CloudTrail log examples, see Example AWS STS API event using the global endpoint in CloudTrail log file.
- Requests made to the AWS STS global endpoint have a value of `us-east-1` for the `aws:RequestedRegion` condition key, regardless of which Region served the request.
- Requests handled by the AWS STS global endpoint do not share a requests per second quota with Regional AWS STS endpoints.

If you have workloads in an opt-in Region and are still using the AWS STS global endpoint, we recommend migrating to AWS STS regional endpoints for improved resiliency and performance. For more information about configuring regional AWS STS endpoints, see AWS STS Regional endpoints in the *AWS SDKs and Tools Reference Guide*.

AWS CloudTrail and Regional endpoints

Calls to regional and global endpoints are logged in the `tlsDetails` field in AWS CloudTrail. Calls to regional endpoints, such as `us-east-2.amazonaws.com`, are logged in CloudTrail to their appropriate region. Calls to the global endpoint, `sts.amazonaws.com`, are logged as calls to a global service. Events for global AWS STS endpoints are logged to us-east-1.

> **Note**
>
> `tlsDetails` can only be viewed for services that support this field. See Services that support TLS details in CloudTrail in the *AWS CloudTrail User Guide*
> For more information, see Logging IAM and AWS STS API calls with AWS CloudTrail.

Enable custom identity broker access to the AWS console

You can write and run code to create a URL that lets users who sign in to your organization's network securely access the AWS Management Console. The URL includes a sign-in token that you get from AWS and that authenticates the user to AWS. The resulting console session might include a distinct `AccessKeyId` due to federation. To trace the access key usage for federation sign-in through related CloudTrail events, see Logging IAM and AWS STS API calls with AWS CloudTrail and AWS Management Console sign-in events.

> **Note**
>
> If your organization uses an identity provider (IdP) that is compatible with SAML, you can set up access to the console without writing code. This works with providers like Microsoft's Active Directory Federation Services or open-source Shibboleth. For details, see Enabling SAML 2.0 federated principals to access the AWS Management Console.

To enable your organization's users to access the AWS Management Console, you can create a custom *identity broker* that performs the following steps:

1. Verify that the user is authenticated by your local identity system.
2. Call the AWS Security Token Service (AWS STS) AssumeRole (recommended) or GetFederationToken API operations to obtain temporary security credentials for the user. To learn about the different methods that you can use to assume a role, see Methods to assume a role. To learn how to pass optional session tags when you obtain your security credentials, see Pass session tags in AWS STS.
   - If you use one of the `AssumeRole*` API operations to get the temporary security credentials for a role, you can include the `DurationSeconds` parameter in your call. This parameter specifies the duration of your role session, from 900 seconds (15 minutes) up to the maximum session duration setting for the role. When you use `DurationSeconds` in an `AssumeRole*` operation, you must call it as an IAM user with long-term credentials. Otherwise, the call to the federation endpoint in step 3 fails. To learn how to view or change the maximum value for a role, see Update the maximum session duration for a role.
   - If you use the `GetFederationToken` API operation to get the credentials, you can include the `DurationSeconds` parameter in your call. This parameter specifies the duration of your role session. The value can range from 900 seconds (15 minutes) to 129,600 seconds (36 hours). You can make this API call only by using the long-term AWS security credentials of an

IAM user. You can also make these calls using AWS account root user credentials, but we do not recommend it. If you make this call as the root user, the default session lasts for one hour. Or you can specify a session from 900 seconds (15 minutes) up to 3,600 seconds (one hour).

3. Call the AWS federation endpoint and supply the temporary security credentials to request a sign-in token.
4. Construct a URL for the console that includes the token:
   - If you use one of the `AssumeRole*` API operations in your URL, you can include the `SessionDuration` HTTP parameter. This parameter specifies the duration of the console session, from 900 seconds (15 minutes) to 43200 seconds (12 hours).
   - If you use the `GetFederationToken` API operation in your URL, you can include the `DurationSeconds` parameter. This parameter specifies the duration of the federated console session. The value can range from 900 seconds (15 minutes) to 129,600 seconds (36 hours).

   Note

   - Your `SessionDuration` cannot be greater than or equal to the maximum session duration setting for the role you're assuming. For example, you've set the maximum session duration for the role you want to assume to 5 hours. Your `SessionDuration` parameter can be 16524 seconds or 4 hours and 59 seconds.
   - Do not use the `SessionDuration` HTTP parameter when you get temporary credentials with `GetFederationToken`. The operation will fail.
   - Using the credentials for one role to assume a different role is called *role chaining*. When you use role chaining, your new credentials are limited to a maximum duration of one hour. When you use roles to grant permissions to applications that run on EC2 instances, those applications are not subject to this limitation.
   - Do not use the `SessionDuration` HTTP parameter when you get temporary credentials through role chaining. The operation will fail.
5. Give the URL to the user or invoke the URL on the user's behalf.

The URL that the federation endpoint provides is valid for 15 minutes after it is created. This differs from the duration (in seconds) of the temporary security credential session that is associated with the URL. Those credentials are valid for the duration you specified when you created them, starting from the time they were created.

Important

The URL grants access to your AWS resources through the AWS Management Console if you have enabled permissions in the associated temporary security credentials. For this reason, you should treat the URL as a secret. We recommend returning the URL through a secure redirect, for example, by using a 302 HTTP response status code over an SSL connection. For more information about the 302 HTTP response status code, see RFC 2616, section 10.3.3.

To complete these tasks, you can use the HTTPS Query API for AWS Identity and Access Management (IAM) and the AWS Security Token Service (AWS STS). Or, you can use programming languages, such as Java, Ruby, or C#, along with the appropriate AWS SDK. Each of these methods is described in the following topics.

Topics

- Example code using IAM query API operations
- Example code using Python
- Example code using Java
- Example showing how to construct the URL (Ruby)

Example code using IAM query API operations

You can construct a URL that gives roles and federated principals direct access to the AWS Management Console. This task uses the IAM and AWS STS HTTPS Query API. For more information about making query requests, see Making Query Requests.

Note

The following procedure contains examples of text strings. To enhance readability, line breaks have been added to some of the longer examples. When you create these strings for your own use, you should omit any line breaks.

To give roles and federated principals access to your resources from the AWS Management Console

1. Authenticate the user in your identity and authorization system.
2. Obtain temporary security credentials for the user. The temporary credentials consist of an access key ID, a secret access key, and a session token. For more information about creating temporary credentials, see Temporary security credentials in IAM.

   To get temporary credentials, you call either the AWS STS AssumeRole API (recommended) or the GetFederationToken API. For more information about the differences between these API operations, see Understanding the API Options for Securely Delegating Access to Your AWS Account in the AWS Security Blog.

   > **Important**
   >
   > When you use the GetFederationToken API to create temporary security credentials, you must specify the permissions that the credentials grant to the user who assumes the role. For any of the API operations that begin with `AssumeRole*`, you use an IAM role to assign permissions. For the other API operations, the mechanism varies with the API. For more details, see Permissions for temporary security credentials. Additionally, if you use the `AssumeRole*` API operations, you must call them as an IAM user with long-term credentials. Otherwise, the call to the federation endpoint in step 3 fails.

3. After you obtain the temporary security credentials, build them into a JSON session string to exchange them for a sign-in token. The following example shows how to encode the credentials. You replace the placeholder text with the appropriate values from the credentials that you receive in the previous step.

   ```
   {"sessionId":"*** temporary access key ID ***",
   "sessionKey":"*** temporary secret access key ***",
   "sessionToken":"*** session token ***"}
   ```

4. URL encode the session string from the previous step. Because the information that you are encoding is sensitive, we recommend that you avoid using a web service for this encoding. Instead, use a locally installed function or feature in your development toolkit to securely encode this information. You can use the `urllib.quote_plus` function in Python, the `URLEncoder.encode` function in Java, or the `CGI.escape` function in Ruby. See the examples later in this topic.

5.

> **Note**
>
> AWS supports POST requests here.

Send your request to the AWS federation endpoint:

`https://region-code.signin.aws.amazon.com/federation`

For a list of possible *region-code* values, see the **Region** column in AWS Sign-In endpoints. You can optionally use the default AWS Sign-In federation endpoint:

`https://signin.aws.amazon.com/federation`

The request must include the `Action` and `Session` parameters, and (optionally) if you used an `AssumeRole*` API operation, a `SessionDuration` HTTP parameter as shown in the following example.

```
Action = getSigninToken
SessionDuration = time in seconds
Session = *** the URL encoded JSON string created in steps 3 & 4 ***
```

> **Note**
>
> The following instructions in this step only work using GET requests.

The `SessionDuration` HTTP parameter specifies the duration of the console session. This is separate from the duration of the temporary credentials that you specify using the `DurationSeconds` parameter. You can specify a `SessionDuration` maximum value of 43,200 (12 hours). If the `SessionDuration` parameter is missing, then the session defaults to the duration of the credentials that you retrieved from AWS STS in step 2 (which defaults to one hour). See the documentation for the `AssumeRole` API for details about how to specify a duration using the `DurationSeconds` parameter. The ability to create a console session that is longer than one hour is intrinsic to the `getSigninToken` operation of the federation endpoint.

Note

- Your `SessionDuration` cannot be greater than or equal to the maximum session duration setting for the role you're assuming. For example, you've set the maximum session duration for the role you want to assume to 5 hours. Your `SessionDuration` parameter can be 16524 seconds or 4 hours and 59 seconds.
- Do not use the `SessionDuration` HTTP parameter when you get temporary credentials with `GetFederationToken`. The operation will fail.
- Using the credentials for one role to assume a different role is called *role chaining*. When you use role chaining, your new credentials are limited to a maximum duration of one hour. When you use roles to grant permissions to applications that run on EC2 instances, those applications are not subject to this limitation.
- Do not use the `SessionDuration` HTTP parameter when you get temporary credentials through role chaining. The operation will fail.

When you enable console sessions with an extended duration, you increase the risk of credential exposure. To help you mitigate this risk, you can immediately disable the active console sessions for any role by choosing **Revoke Sessions** on the **Role Summary** IAM console page. For more information, see Revoke IAM role temporary security credentials.

The following is an example of what your request might look like. The lines are wrapped here for readability, but you should submit it as a one-line string.

```
https://signin.aws.amazon.com/federation
?Action=getSigninToken
&SessionDuration=1800
&Session=%7B%22sessionId%22%3A+%22ASIAJUMHIZPTOKTBMK5A%22%2C+%22sessionKey%22
%3A+%22LSD7LWI%2FL%2FN%2BgYpan5QFz0XUpc8s7HYjRsgcsrsm%22%2C+%22sessionToken%2
2%3A+%22FQoDYXdzEBQaDLbj3VWv2u50NN%2F3yyLSASwYtWhPnGPMNmzZFfZsL0Qd3vtYHw5A5dW
AjOsrkdPkghomIe3mJip5%2F0djDBbo7SmO%2FENDEiCdpsQKodTpleKA8xQq0CwFg6a69xdEBQT8
FipATnLbKoyS4b%2FebhnsTUjZZQWp0wXXqFF7gSm%2FMe2tXe0jzsdP0O12obez9lijPSdF1k2b5
PfGhiuyAR9aD5%2BubM0pY86fKex1qsytjvyTbZ9nXe6DvxVDcnCOhOGETJ7XFkSFdH0v%2FYR25C
UAhJ3nXIkIbG7Ucv9cOEpCf%2Fg23ijRgILIBQ%3D%3D%22%7D
```

The response from the federation endpoint is a JSON document with a `SigninToken` value. It will look similar to the following example.

```
{"SigninToken":"*** the SigninToken string ***"}
```

6.

> **Note**
>
> AWS supports POST requests here.

Finally, create the URL that your users can use to access the AWS Management Console. The URL is the same federation URL endpoint that you used in Step 5, plus the following parameters:

```
?Action = login
&Issuer = *** the form-urlencoded URL for your internal sign-in page ***
&Destination = *** the form-urlencoded URL to the desired AWS console page ***
&SigninToken = *** the value of SigninToken received in the previous step ***
```

> **Note**
>
> The following instructions in this step only work using GET API.

The following example shows what the final URL might look like. The URL is valid for 15 minutes from the time it is created. The temporary security credentials and console session embedded within the URL are valid for the duration you specify in the `SessionDuration` HTTP parameter when you initially request them.

```
https://signin.aws.amazon.com/federation
?Action=login
&Issuer=https%3A%2F%2Fexample.com
&Destination=https%3A%2F%2Fconsole.aws.amazon.com%2F
&SigninToken=VCQgs5qZZt3Q6fn8Tr5EXAMPLEmLnwB7JjUc-SHwnUUWabcRdnWsi4DBn-dvC
CZ85wrD0nmldUcZEXAMPLE-vXYH4Q__mleuF_W2BE5HYexbe9y4Of-kje53SsjNNecATfjIzpW1
WibbnH6YcYRiBoffZBGExbEXAMPLE5aiKX4THWjQKC6gg6alHu6JFrnOJoK3dtP6I9a6hi6yPgm
iOkPZMmNGmhsvVxetKzr8mx3pxhHbMEXAMPLETv1pij0rok3IyCR2YVcIjqwfWv32HU2Xlj471u
3fU6uOfUComeKiqTGX974xzJOZbdmX_t_lLrhEXAMPLEDDIisSnyHGw2xaZZqudm4mo2uTDk9Pv
9l5K0ZCqIgEXAMPLEcA6tgLPykEWGUyH6BdSC6166n4M4JkXIQgac7_7821YqixsNxZ6rsrpzwf
nQoS14O7R0eJCCJ684EXAMPLEZRdBNnuLbUYpz2Iw3vIN0tQgOujwnwydPscM9F7foaEK3jwMkg
Apeb1-6L_OB12MZhuFxx55555EXAMPLEhyETEd4ZulKPdXHkgl6T9ZkIlHz2Uy1RUTUhhUxNtSQ
nWc5xkbBoEcXqpoSIeK7yhje9Vzhd61AEXAMPLElbWeouACEMG6-Vd3dAgFYd6i5FYoyFrZLWvm
```

```
0LSG7RyYKeYN5VIzUk3YWQpyjP0RiT5KUrsUi-NEXAMPLExMOMdoODBEgKQsk-iu2ozh6r8bxwC
RNhujg
```

Example code using Python

The following examples show how to use Python to programmatically construct a URL that gives users direct access to the AWS Management Console. There are two examples:

- Federate via GET requests to AWS
- Federate via POST requests to AWS

Both examples use the AWS SDK for Python (Boto3) and AssumeRole API to obtain temporary security credentials.

Do not include `SessionDuration` if your `AssumeRoleSession` credentials are from role chaining. If you include `SessionDuration`, the operation will fail.

Use GET Requests

```
import urllib, json, sys
import requests # 'pip install requests'
import boto3 # AWS SDK for Python (Boto3) 'pip install boto3'

# Step 1: Authenticate user in your own identity system.

# Step 2: Using the access keys for an IAM user in your AWS account,
# call "AssumeRole" to get temporary access keys for the role or federated principal

# Note: Calls to AWS STS AssumeRole must be signed using the access key ID
# and secret access key of an IAM user or using existing temporary credentials.
# The credentials can be in Amazon EC2 instance metadata, in environment variables,
# or in a configuration file, and will be discovered automatically by the
# client('sts') function. For more information, see the Python SDK docs:
# http://boto3.readthedocs.io/en/latest/reference/services/sts.html
# http://boto3.readthedocs.io/en/latest/reference/services/
sts.html#STS.Client.assume_role
sts_connection = boto3.client('sts')

assumed_role_object = sts_connection.assume_role(
    RoleArn="arn:aws:iam::account-id:role/ROLE-NAME",
    RoleSessionName="AssumeRoleSession",
```

```
)

# Step 3: Format resulting temporary credentials into JSON
url_credentials = {}
url_credentials['sessionId'] =
 assumed_role_object.get('Credentials').get('AccessKeyId')
url_credentials['sessionKey'] =
 assumed_role_object.get('Credentials').get('SecretAccessKey')
url_credentials['sessionToken'] =
 assumed_role_object.get('Credentials').get('SessionToken')
json_string_with_temp_credentials = json.dumps(url_credentials)

# Step 4. Make request to AWS federation endpoint to get sign-in token. Construct the
 parameter string with
# the sign-in action request, a 12-hour session duration, and the JSON document with
 temporary credentials
# as parameters.
request_parameters = "?Action=getSigninToken"
request_parameters += "&SessionDuration=43200"
if sys.version_info[0] < 3:
    def quote_plus_function(s):
        return urllib.quote_plus(s)
else:
    def quote_plus_function(s):
        return urllib.parse.quote_plus(s)
request_parameters += "&Session=" +
 quote_plus_function(json_string_with_temp_credentials)
request_url = "https://signin.aws.amazon.com/federation" + request_parameters
r = requests.get(request_url)
# Returns a JSON document with a single element named SigninToken.
signin_token = json.loads(r.text)

# Step 5: Create URL where users can use the sign-in token to sign in to
# the console. This URL must be used within 15 minutes after the
# sign-in token was issued.
request_parameters = "?Action=login"
request_parameters += "&Issuer=Example.org"
request_parameters += "&Destination=" + quote_plus_function("https://
console.aws.amazon.com/")
request_parameters += "&SigninToken=" + signin_token["SigninToken"]
request_url = "https://signin.aws.amazon.com/federation" + request_parameters

# Send final URL to stdout
```

```
print (request_url)
```

Use POST Requests

```
import urllib, json, sys
import requests # 'pip install requests'
import boto3 # AWS SDK for Python (Boto3) 'pip install boto3'
import os
from selenium import webdriver # 'pip install selenium', 'brew install chromedriver'

# Step 1: Authenticate user in your own identity system.

# Step 2: Using the access keys for an IAM user in your AAWS account,
# call "AssumeRole" to get temporary access keys for the role or federated principal

# Note: Calls to AWS STS AssumeRole must be signed using the access key ID
# and secret access key of an IAM user or using existing temporary credentials.
# The credentials can be in Amazon EC2 instance metadata, in environment variables,

# or in a configuration file, and will be discovered automatically by the
# client('sts') function. For more information, see the Python SDK docs:
# http://boto3.readthedocs.io/en/latest/reference/services/sts.html
# http://boto3.readthedocs.io/en/latest/reference/services/
sts.html#STS.Client.assume_role
if sys.version_info[0] < 3:
    def quote_plus_function(s):
        return urllib.quote_plus(s)
else:
    def quote_plus_function(s):
        return urllib.parse.quote_plus(s)

sts_connection = boto3.client('sts')

assumed_role_object = sts_connection.assume_role(
    RoleArn="arn:aws:iam::account-id:role/ROLE-NAME",
    RoleSessionName="AssumeRoleDemoSession",
)

# Step 3: Format resulting temporary credentials into JSON
url_credentials = {}
url_credentials['sessionId'] =
 assumed_role_object.get('Credentials').get('AccessKeyId')
```

```
url_credentials['sessionKey'] =
 assumed_role_object.get('Credentials').get('SecretAccessKey')
url_credentials['sessionToken'] =
 assumed_role_object.get('Credentials').get('SessionToken')
json_string_with_temp_credentials = json.dumps(url_credentials)

# Step 4. Make request to AWS federation endpoint to get sign-in token. Construct the
 parameter string with
# the sign-in action request, a 12-hour session duration, and the JSON document with
 temporary credentials
# as parameters.
request_parameters = {}
request_parameters['Action'] = 'getSigninToken'
request_parameters['SessionDuration'] = '43200'
request_parameters['Session'] = json_string_with_temp_credentials

request_url = "https://signin.aws.amazon.com/federation"
r = requests.post( request_url, data=request_parameters)

# Returns a JSON document with a single element named SigninToken.
signin_token = json.loads(r.text)

# Step 5: Create a POST request where users can use the sign-in token to sign in to
# the console. The POST request must be made within 15 minutes after the
# sign-in token was issued.
request_parameters = {}
request_parameters['Action'] = 'login'
request_parameters['Issuer']='Example.org'
request_parameters['Destination'] = 'https://console.aws.amazon.com/'
request_parameters['SigninToken'] =signin_token['SigninToken']

jsrequest = '''
var form = document.createElement('form');
form.method = 'POST';
form.action = '{request_url}';
request_parameters = {request_parameters}
for (var param in request_parameters) {{
    if (request_parameters.hasOwnProperty(param)) {{
        const hiddenField = document.createElement('input');
        hiddenField.type = 'hidden';
        hiddenField.name = param;
        hiddenField.value = request_parameters[param];
        form.appendChild(hiddenField);
    }}
```

```
}}
document.body.appendChild(form);
form.submit();
'''.format(request_url=request_url, request_parameters=request_parameters)

driver = webdriver.Chrome()
driver.execute_script(jsrequest)
input("Press Enter to close the browser window...")
```

Example code using Java

The following example shows how to use Java to programmatically construct a URL that gives users direct access to the AWS Management Console. The following code snippet uses the AWS SDK for Java.

```
import java.net.URLEncoder;
import java.net.URL;
import java.net.URLConnection;
import java.io.BufferedReader;
import java.io.InputStreamReader;
// Available at http://www.json.org/java/index.html
import org.json.JSONObject;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;


/* Calls to AWS STS API operations must be signed using the access key ID
   and secret access key of an IAM user or using existing temporary
   credentials. The credentials should not be embedded in code. For
   this example, the code looks for the credentials in a
   standard configuration file.
*/
AWSCredentials credentials =
  new PropertiesCredentials(
         AwsConsoleApp.class.getResourceAsStream("AwsCredentials.properties"));

AWSSecurityTokenServiceClient stsClient =
  new AWSSecurityTokenServiceClient(credentials);
```

```
GetFederationTokenRequest getFederationTokenRequest =
  new GetFederationTokenRequest();
getFederationTokenRequest.setDurationSeconds(1800);
getFederationTokenRequest.setName("UserName");

// A sample policy for accessing Amazon Simple Notification Service (Amazon SNS) in the
 console.

String policy = "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Action\":\"sns:*
\"," +
  "\"Effect\":\"Allow\",\"Resource\":\"*\"}]}";

getFederationTokenRequest.setPolicy(policy);

GetFederationTokenResult federationTokenResult =
  stsClient.getFederationToken(getFederationTokenRequest);

Credentials federatedCredentials = federationTokenResult.getCredentials();

// The issuer parameter specifies your internal sign-in
// page, for example https://mysignin.internal.mycompany.com/.
// The console parameter specifies the URL to the destination console of the
// AWS Management Console. This example goes to Amazon SNS.
// The signin parameter is the URL to send the request to.

String issuerURL = "https://mysignin.internal.mycompany.com/";
String consoleURL = "https://console.aws.amazon.com/sns";
String signInURL = "https://signin.aws.amazon.com/federation";

// Create the sign-in token using temporary credentials,
// including the access key ID,  secret access key, and session token.
String sessionJson = String.format(
  "{\"%1$s\":\"%2$s\",\"%3$s\":\"%4$s\",\"%5$s\":\"%6$s\"}",
  "sessionId", federatedCredentials.getAccessKeyId(),
  "sessionKey", federatedCredentials.getSecretAccessKey(),
  "sessionToken", federatedCredentials.getSessionToken());

// Construct the sign-in request with the request sign-in token action, a
// 12-hour console session duration, and the JSON document with temporary
// credentials as parameters.

String getSigninTokenURL = signInURL +
                           "?Action=getSigninToken" +
                           "&DurationSeconds=43200" +
```

```
                            "&SessionType=json&Session=" +
                            URLEncoder.encode(sessionJson,"UTF-8");

URL url = new URL(getSigninTokenURL);

// Send the request to the AWS federation endpoint to get the sign-in token
URLConnection conn = url.openConnection ();

BufferedReader bufferReader = new BufferedReader(new
  InputStreamReader(conn.getInputStream()));
String returnContent = bufferReader.readLine();

String signinToken = new JSONObject(returnContent).getString("SigninToken");

String signinTokenParameter = "&SigninToken=" + URLEncoder.encode(signinToken,"UTF-8");

// The issuer parameter is optional, but recommended. Use it to direct users
// to your sign-in page when their session expires.

String issuerParameter = "&Issuer=" + URLEncoder.encode(issuerURL, "UTF-8");

// Finally, present the completed URL for the AWS console session to the user

String destinationParameter = "&Destination=" + URLEncoder.encode(consoleURL,"UTF-8");
String loginURL = signInURL + "?Action=login" +
                     signinTokenParameter + issuerParameter + destinationParameter;
```

Example showing how to construct the URL (Ruby)

The following example shows how to use Ruby to programmatically construct a URL that gives users direct access to the AWS Management Console. This code snippet uses the AWS SDK for Ruby.

```
require 'rubygems'
require 'json'
require 'open-uri'
require 'cgi'
require 'aws-sdk'

# Create a new STS instance
#
# Note: Calls to AWS STS API operations must be signed using an access key ID
# and secret access key. The credentials can be in EC2 instance metadata
# or in environment variables and will be automatically discovered by
```

```
# the default credentials provider in the AWS Ruby SDK.
sts = Aws::STS::Client.new()

# The following call creates a temporary session that returns
# temporary security credentials and a session token.
# The policy grants permissions to work
# in the AWS SNS console.

session = sts.get_federation_token({
  duration_seconds: 1800,
  name: "UserName",
  policy: "{\"Version\":\"2012-10-17\",       \"Statement\":{\"Effect\":\"Allow\",
\"Action\":\"sns:*\",\"Resource\":\"*\"}}",
})

# The issuer value is the URL where users are directed (such as
# to your internal sign-in page) when their session expires.
#
# The console value specifies the URL to the destination console.
# This example goes to the Amazon SNS console.
#
# The sign-in value is the URL of the AWS STS federation endpoint.
issuer_url = "https://mysignin.internal.mycompany.com/"
console_url = "https://console.aws.amazon.com/sns"
signin_url = "https://signin.aws.amazon.com/federation"

# Create a block of JSON that contains the temporary credentials
# (including the access key ID, secret access key, and session token).
session_json = {
  :sessionId => session.credentials[:access_key_id],
  :sessionKey => session.credentials[:secret_access_key],
  :sessionToken => session.credentials[:session_token]
}.to_json

# Call the federation endpoint, passing the parameters
# created earlier and the session information as a JSON block.
# The request returns a sign-in token that's valid for 15 minutes.
# Signing in to the console with the token creates a session
# that is valid for 12 hours.
get_signin_token_url = signin_url +
                       "?Action=getSigninToken" +
                       "&SessionType=json&Session=" +
                       CGI.escape(session_json)
```

```
returned_content = URI.parse(get_signin_token_url).read

# Extract the sign-in token from the information returned
# by the federation endpoint.
signin_token = JSON.parse(returned_content)['SigninToken']
signin_token_param = "&SigninToken=" + CGI.escape(signin_token)

# Create the URL to give to the user, which includes the
# sign-in token and the URL of the console to open.
# The "issuer" parameter is optional but recommended.
issuer_param = "&Issuer=" + CGI.escape(issuer_url)
destination_param = "&Destination=" + CGI.escape(console_url)
login_url = signin_url + "?Action=login" + signin_token_param +
  issuer_param + destination_param
```

Tags for AWS Identity and Access Management resources

A *tag* is a custom attribute label that you can assign to an AWS resource. Each tag has two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, `Project`, or `Purpose`).
- An optional field known as a *tag value* (for example, `111122223333`, `Production`, or a team name). Omitting the tag value is the same as using an empty string.

Together these are known as key-value pairs. For limits on the number of tags you can have on IAM resources, see IAM and AWS STS quotas.

> **Note**
>
> For details about case sensitivity for tag keys and tag key values, see Case sensitivity.

Tags help you identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to an IAM role that you assign to an Amazon S3 bucket. For more information about tagging strategies, see the *Tagging AWS resources User Guide*.

In addition to identifying, organizing, and tracking your IAM resources with tags, you can use tags in IAM policies to help control who can view and interact with your resources. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see Pass session tags in AWS STS.

Topics

- Choose an AWS tag naming convention
- Rules for tagging in IAM and AWS STS
- Tag IAM users
- Tag IAM roles
- Tag customer managed policies
- Tag OpenID Connect (OIDC) identity providers
- Tag IAM SAML identity providers
- Tag instance profiles for Amazon EC2 roles
- Tag server certificates
- Tag virtual MFA devices
- Tag account access manager instances
- Pass session tags in AWS STS

Choose an AWS tag naming convention

When you begin attaching tags to your IAM resources, choose your tag naming convention carefully. Apply the same convention to all of your AWS tags. This is especially important if you use tags in policies to control access to AWS resources. If you already use tags in AWS, review your naming convention and adjust it accordingly.

> **Note**
>
> If your account is a member of AWS Organizations, see Tag policies in the AWS Organizations user guide to learn more about using tags in AWS Organizations.

Best practices for tag naming

These are some best practices and naming conventions for tags.

Ensure that tag names are used consistently. For example, the tags `CostCenter` and `costcenter` are different, so one might be configured as a cost allocation tag for financial analysis and

reporting and the other one might not be. Similarly, the `Name` tag appears in the AWS Console for many resources, but the `name` tag does not. For details about case sensitivity for tag keys and tag key values, see Case sensitivity.

A number of tags are predefined by AWS or created automatically by various AWS services. Many AWS-defined tags names use all lowercase, with hyphens separating words in the name, and prefixes to identify the source service for the tag. For example:

- `aws:ec2spot:fleet-request-id` identifies the Amazon EC2 Spot Instance Request that launched the instance.
- `aws:cloudformation:stack-name` identifies the CloudFormation stack that created the resource.
- `elasticbeanstalk:environment-name` identifies the application that created the resource.

Consider naming your tags using all lowercase, with hyphens separating words, and a prefix identifying the organization name or abbreviated name. For example, for a fictitious company named *AnyCompany*, you might define tags such as:

- `anycompany:cost-center` to identify the internal Cost Center code
- `anycompany:environment-type` to identify whether the environment is development, test, or production
- `anycompany:application-id` to identify the application the resource was created for

The prefix ensures that tags are clearly identified as having been defined by your organization and not by AWS or a third-party tool that you may be using. Using all lowercase with hyphens for separators avoids confusion about how to capitalize a tag name. For example, `anycompany:project-id` is simpler to remember than `ANYCOMPANY:ProjectID`, `anycompany:projectID`, or `Anycompany:ProjectId`.

Rules for tagging in IAM and AWS STS

A number of conventions govern the creation and application of tags in IAM and AWS STS.

Naming tags

Observe the following conventions when formulating a tag naming convention for IAM resources, AWS STS assume-role sessions, and AWS STS federated user sessions:

Character requirements – Tag keys and values can include any combination of letters, numbers, spaces, and _ . : / = + - @ symbols.

Case sensitivity – Case sensitivity for tag keys differs depending on the type of IAM resource that is tagged. Tag key values for IAM users and roles are not case sensitive, but case is preserved. This means that you cannot have separate **`Department`** and **`department`** tag keys. If you have tagged a user with the **`Department=finance`** tag and you add the **`department=hr`** tag, it replaces the first tag. A second tag is not added.

For other IAM resource types, tag key values are case sensitive. That means you can have separate **`Costcenter`** and **`costcenter`** tag keys. For example, if you have tagged a customer managed policy with the **`Costcenter = 1234`** tag and you add the **`costcenter = 5678`** tag, the policy will have both the **`Costcenter`** and **`costcenter`** tag keys.

As a best practice, we recommend that you avoid using similar tags with inconsistent case treatment. We recommend that you decide on a strategy for capitalizing tags, and consistently implement that strategy across all resource types. To learn more about best practices for tagging, see Tagging AWS Resources in the AWS General Reference.

The following lists show the differences in case sensitivity for tag keys that are attached to IAM resources.

Tag key values are **not** case sensitive:

- IAM roles
- IAM users

Tag key values are case sensitive:

- Customer managed policies
- Instance profiles
- OpenID Connect identity providers
- SAML identity providers
- Server certificates
- Virtual MFA devices

Additionally, the following rules apply:

- You cannot create a tag key or value that begins with the text **`aws:`**. This tag prefix is reserved for AWS internal use.
- You can create a tag with an empty value such as **`phoneNumber =`** . You cannot create an empty tag key.
- You cannot specify multiple values in a single tag, but you can create a custom multivalue structure in the single value. For example, assume that the user Zhang works on the engineering team and the QA team. If you attach the **`team = Engineering`** tag and then attach the **`team = QA`** tag, you change the value of the tag from **`Engineering`** to **`QA`**. Instead, you can include multiple values in a single tag with a custom separator. In this example, you could attach the **`team = Engineering:QA`** tag to Zhang.

> **Note**
>
> To control access to engineers in this example using the **`team`** tag, you must create a policy that allows for every configuration that might include **`Engineering`**, including **`Engineering:QA`**. To learn more about using tags in policies, see Controlling access to and for IAM users and roles using tags.

Applying and editing tags

Observe the following conventions when attaching tags to IAM resources:

- You can tag most IAM resources, but not groups, assumed roles, access reports, or hardware-based MFA devices.
- You cannot use Tag Editor to tag IAM resources. Tag Editor does not support IAM tags. For information about using Tag Editor with other services, see Working with Tag Editor in the *AWS Resource Groups User Guide*.
- To tag an IAM resource, you must have specific permissions. To tag or untag resources, you must also have permission to list tags. For more information, see the list of topics for each IAM resource at the end of this page.
- The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.
- You can apply the same tag to multiple IAM resources. For example, suppose you have a department named `AWS_Development` with 12 members. You can have 12 users and a role with the tag key of **`department`** and a value of **`awsDevelopment`** (**`department =`**

awsDevelopment). You can also use the same tag on resources in other services that support tagging.

- IAM entities (users or roles) cannot have multiple instances of the same tag key. For example, if you have a user with the tag key-value pair **costCenter = 1234**, you can then attach the tag key-value pair **costCenter = 5678**. IAM updates the value of the **costCenter** tag to **5678**.
- To edit a tag that is attached to an IAM entity (user or role), attach a tag with a new value to overwrite the existing tag. For example, assume that you have a user with the tag key-value pair **department = Engineering**. If you need to move the user to the QA department, then you can attach the **department = QA** tag key-value pair to the user. This results in the **Engineering** value of the **department** tag key being replaced with the **QA** value.

Tag IAM users

You can use IAM tag key-value pairs to add custom attributes to an IAM user. For example, to add location information to a user, you can add the tag key **location** and the tag value **us_wa_seattle**. Or you could use three separate location tag key-value pairs: **loc-country = us**, **loc-state = wa**, and **loc-city = seattle**. You can use tags to control a user's access to resources or to control what tags can be attached to a user. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see Pass session tags in AWS STS.

Permissions required for tagging IAM users

You must configure permissions to allow an IAM user to tag other users. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListUserTags`
- `iam:TagUser`
- `iam:UntagUser`

To allow an IAM user to add, list, or remove a tag for a specific user

Add the following statement to the permissions policy for the IAM user that needs to manage tags. Use your account number and replace *<username>* with the name of the user whose tags need

to be managed. To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListUserTags",
        "iam:TagUser",
        "iam:UntagUser"
    ],
    "Resource": "arn:aws:iam::<account-number>:user/<username>"
}
```

To allow an IAM user to self-manage tags

Add the following statement to the permissions policy for users to allow users to manage their own tags. To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListUserTags",
        "iam:TagUser",
        "iam:UntagUser"
    ],
    "Resource": "arn:aws:iam::user/${aws:username}"
}
```

To allow an IAM user to add a tag to a specific user

Add the following statement to the permissions policy for the IAM user that needs to add, but not remove, tags for a specific user.

> **Note**
>
> The `iam:TagUser` action requires that you also include the `iam:ListUserTags` action.

To use this policy, replace *<username>* with the name of the user whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListUserTags",
        "iam:TagUser"
    ],
    "Resource": "arn:aws:iam::<account-number>:user/<username>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on IAM users (console)

You can manage tags for IAM users from the AWS Management Console.

To manage tags on users (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Users** and then choose the name of the user that you want to edit.
3. Choose the **Tags** tab and then complete one of the following actions:
   - Choose **Add new tag** if the user does not yet have tags.
   - Choose **Manage tags** to manage the existing set of tags.
4. Add or remove tags to complete the set of tags. Then choose **Save changes**.

Managing tags on IAM users (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM users. You can use the AWS CLI or the AWS API to manage tags for IAM users.

To list the tags currently attached to an IAM user (AWS CLI or AWS API)

- AWS CLI: aws iam list-user-tags

- AWS API: ListUserTags

To attach tags to an IAM user (AWS CLI or AWS API)

- AWS CLI: aws iam tag-user
- AWS API: TagUser

To remove tags from an IAM user (AWS CLI or AWS API)

- AWS CLI: aws iam untag-user
- AWS API: UntagUser

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag IAM roles

You can use IAM tag key-value pairs to add custom attributes to an IAM role. For example, to add location information to a role, you can add the tag key **`location`** and the tag value **`us_wa_seattle`**. Or you could use three separate location tag key-value pairs: **`loc-country = us`**, **`loc-state = wa`**, and **`loc-city = seattle`**. You can use tags to control a role's access to resources or to control what tags can be attached to a role. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see Pass session tags in AWS STS.

Permissions required for tagging IAM roles

You must configure permissions to allow an IAM role to tag other entities (users or roles). You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListRoleTags`
- `iam:TagRole`
- `iam:UntagRole`

- `iam:ListUserTags`
- `iam:TagUser`
- `iam:UntagUser`

To allow an IAM role to add, list, or remove a tag for a specific user

Add the following statement to the permissions policy for the IAM role that needs to manage tags. Use your account number and replace *<username>* with the name of the user whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListUserTags",
        "iam:TagUser",
        "iam:UntagUser"
    ],
    "Resource": "arn:aws:iam::<account-number>:user/<username>"
}
```

To allow an IAM role to add a tag to a specific user

Add the following statement to the permissions policy for the IAM role that needs to add, but not remove, tags for a specific user.

To use this policy, replace *<username>* with the name of the user whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListUserTags",
        "iam:TagUser"
    ],
    "Resource": "arn:aws:iam::<account-number>:user/<username>"
}
```

To allow an IAM role to add, list, or remove a tag for a specific role

Add the following statement to the permissions policy for the IAM role that needs to manage tags. Replace *<rolename>* with the name of the role whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListRoleTags",
        "iam:TagRole",
        "iam:UntagRole"
    ],
    "Resource": "arn:aws:iam::<account-number>:role/<rolename>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on IAM roles (console)

You can manage tags for IAM roles from the AWS Management Console.

To manage tags on roles (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Roles** and then choose the name of the role that you want to edit.
3. Choose the **Tags** tab and then complete one of the following actions:
   - Choose **Add new tag** if the role does not yet have tags.
   - Choose **Manage tags** to manage the existing set of tags.
4. Add or remove tags to complete the set of tags. Then, choose **Save changes**.

Managing tags on IAM roles (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM roles. You can use the AWS CLI or the AWS API to manage tags for IAM roles.

To list the tags currently attached to an IAM role (AWS CLI or AWS API)

- AWS CLI: aws iam list-role-tags
- AWS API: ListRoleTags

To attach tags to an IAM role (AWS CLI or AWS API)

- AWS CLI: aws iam tag-role
- AWS API: TagRole

To remove tags from an IAM role (AWS CLI or AWS API)

- AWS CLI: aws iam untag-role
- AWS API: UntagRole

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag customer managed policies

You can use IAM tag key-value pairs to add custom attributes to your customer managed policies. For example, to tag a policy with department information, you can add the tag key **`Department`** and the tag value **`eng`**. Or, you might want to tag policies to indicate that they are for a specific environment, such as **`Environment = lab`**. You can use tags to control access to resources or to control what tags can be attached to a resource. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see Pass session tags in AWS STS.

Permissions required for tagging customer managed policies

You must configure permissions to allow an IAM entity (users or roles) to tag customer managed policies. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListPolicyTags`
- `iam:TagPolicy`
- `iam:UntagPolicy`

To allow an IAM entity (user or role) to add, list, or remove a tag for a customer managed policy

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace *<policyname>* with the name of the policy whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListPolicyTags",
        "iam:TagPolicy",
        "iam:UntagPolicy"
    ],
    "Resource": "arn:aws:iam::<account-number>:policy/<policyname>"
}
```

To allow an IAM entity (user or role) to add a tag to a specific customer managed policy

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific policy.

> **Note**
>
> The `iam:TagPolicy` action requires that you also include the `iam:ListPolicyTags` action.

To use this policy, replace *<policyname>* with the name of the policy whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
```

```
    "Action": [
        "iam:ListPolicyTags",
        "iam:TagPolicy"
    ],
    "Resource": "arn:aws:iam::<account-number>:policy/<policyname>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on IAM customer managed policies (console)

You can manage tags for IAM customer managed policies from the AWS Management Console.

To manage tags on customer managed policies (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Policies** and then choose the name of the customer managed policy that you want to edit.
3. Choose the **Tags** tab and then choose **Manage tags**.
4. Add or remove tags to complete the set of tags. Then choose **Save changes**.

Managing tags on IAM customer managed policies (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM customer managed policies. You can use the AWS CLI or the AWS API to manage tags for IAM customer managed policies.

To list the tags currently attached to an IAM customer managed policy (AWS CLI or AWS API)

- AWS CLI: aws iam list-policy-tags
- AWS API: ListPolicyTags

To attach tags to an IAM customer managed policy(AWS CLI or AWS API)

- AWS CLI: aws iam tag-policy
- AWS API: TagPolicy

To remove tags from an IAM customer managed policy (AWS CLI or AWS API)

- AWS CLI: aws iam untag-policy
- AWS API: UntagPolicy

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag OpenID Connect (OIDC) identity providers

You can use IAM tag key-values to add custom attributes to IAM OpenID Connect (OIDC) identity providers. For example, to identify an OIDC identity provider, you can add the tag key **`google`** and the tag value **`oidc`**. You can use tags to control access to resources or to control what tags can be attached to an object. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

Permissions required for tagging IAM OIDC identity providers

You must configure permissions to allow an IAM entity (user or role) to tag IAM OIDC identity providers. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListOpenIDConnectProviderTags`
- `iam:TagOpenIDConnectProvider`
- `iam:UntagOpenIDConnectProvider`

To allow an IAM entity to add, list, or remove a tag for an IAM OIDC identity provider

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace *`<OIDCProviderName>`* with the name of the OIDC provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListOpenIDConnectProviderTags",
```

```
        "iam:TagOpenIDConnectProvider",
        "iam:UntagOpenIDConnectProvider"
    ],
    "Resource": "arn:aws:iam::<account-number>:oidc-provider/<OIDCProviderName>"
}
```

To allow an IAM entity (user or role) to add a tag to a specific IAM OIDC identity provider

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific identity provider.

> **Note**
>
> The `iam:TagOpenIDConnectProvider` action requires that you also include the `iam:ListOpenIDConnectProviderTags` action.

To use this policy, replace *<OIDCProviderName>* with the name of the OIDC provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListOpenIDConnectProviderTags",
        "iam:TagOpenIDConnectProvider"
    ],
    "Resource": "arn:aws:iam::<account-number>:oidc-provider/<OIDCProviderName>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on IAM OIDC identity providers (console)

You can manage tags for IAM OIDC identity providers from the AWS Management Console.

To manage tags on OIDC identity providers (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane of the console, choose **Identity providers** and then choose the name of the identity provider that you want to edit.
3. Choose the **Tags** tab, then in the **Tags** section, choose **Manage tags** and then complete one of the following actions:
   - Choose **Add tag** if the OIDC identity provider does not yet have tags or to add a new tag.
   - Edit existing tag keys and values.
   - Choose **Remove tag** to remove a tag.
4. Then choose **Save changes**.

Managing tags on IAM OIDC identity providers (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM OIDC identity providers. You can use the AWS CLI or the AWS API to manage tags for IAM OIDC identity providers.

To list the tags currently attached to an IAM OIDC identity provider (AWS CLI or AWS API)

- AWS CLI: aws iam list-open-id-connect-provider-tags
- AWS API: ListOpenIDConnectProviderTags

To attach tags to an IAM OIDC identity provider (AWS CLI or AWS API)

- AWS CLI: aws iam tag-open-id-connect-provider
- AWS API: TagOpenIDConnectProvider

To remove tags from an IAM OIDC identity provider (AWS CLI or AWS API)

- AWS CLI: aws iam untag-open-id-connect-provider
- AWS API: UntagOpenIDConnectProvider

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag IAM SAML identity providers

You can use IAM tag key-value pairs to add custom attributes to SAML identity providers. For example, to identify a provider, you can add the tag key **`okta`** and the tag value **`saml`**. You can use tags to control access to resources or to control what tags can be attached to an object. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

Permissions required for tagging SAML identity providers

You must configure permissions to allow an IAM entity (users or roles) to tag SAML 2.0–based Identity Providers (IdPs). You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListSAMLProviderTags`
- `iam:TagSAMLProvider`
- `iam:UntagSAMLProvider`

To allow an IAM entity (user or role) to add, list, or remove a tag for a SAML identity provider

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace *<SAMLProviderName>* with the name of the SAML provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListSAMLProviderTags",
        "iam:TagSAMLProvider",
        "iam:UntagSAMLProvider"
    ],
    "Resource": "arn:aws:iam::<account-number>:saml-provider/<SAMLProviderName>"
}
```

To allow an IAM entity (user or role) to add a tag to a specific SAML identity provider

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific SAML provider.

> **Note**
>
> The `iam:TagSAMLProvider` action requires that you also include the `iam:ListSAMLProviderTags` action.

To use this policy, replace *<SAMLProviderName>* with the name of the SAML provider whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListSAMLProviderTags",
        "iam:TagSAMLProvider"
    ],
    "Resource": "arn:aws:iam::<account-number>:saml-provider/<SAMLProviderName>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on IAM SAML identity providers (console)

You can manage tags for IAM SAML Identity Providers from the AWS Management Console.

To manage tags on SAML identity providers (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Identity providers** and then choose the name of the SAML identity provider that you want to edit.
3. Choose the **Tags** tab, then in the **Tags** section, choose **Manage tags** and then complete one of the following actions:
   - Choose **Add tag** if the SAML identity provider does not yet have tags or to add a new tag.
   - Edit existing tag keys and values.
   - Choose **Remove tag** to remove a tag.

4. Add or remove tags to complete the set of tags. Then choose **Save changes**.

Managing tags on IAM SAML identity providers (AWS CLI or AWS API)

You can list, attach, or remove tags for IAM SAML identity providers. You can use the AWS CLI or the AWS API to manage tags for IAM SAML identity providers.

To list the tags currently attached to an SAML identity provider (AWS CLI or AWS API)

- AWS CLI: aws iam list-saml-provider-tags
- AWS API: ListSAMLProviderTags

To attach tags to a SAML identity provider (AWS CLI or AWS API)

- AWS CLI: aws iam tag-saml-provider
- AWS API: TagSAMLProvider

To remove tags from a SAML identity provider (AWS CLI or AWS API)

- AWS CLI: aws iam untag-saml-provider
- AWS API: UntagSAMLProvider

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag instance profiles for Amazon EC2 roles

When you launch an Amazon EC2 instance, you specify an IAM role to associate with the instance. An instance profile is a container for an IAM role that you can use to pass role information to an Amazon EC2 instance when the instance starts. You can tag instance profiles when you use the AWS CLI or AWS API.

You can use IAM tag key-value pairs to add custom attributes to an instance profile. For example, to add department information to an instance profile, you can add the tag key **`access-team`** and

the tag value **eng**. Doing this gives principals with matching tags access to instance profiles with the same tag. You could use multiple tag key-value pairs to specify a team and project: **access-team = eng** , and **project = peg**. You can use tags to control a user's access to resources or to control what tags can be attached to a user. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see Pass session tags in AWS STS.

Permissions required for tagging instance profiles

You must configure permissions to allow an IAM entity (user or role) to tag instance profiles. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListInstanceProfileTags`
- `iam:TagInstanceProfile`
- `iam:UntagInstanceProfile`

To allow an IAM entity (user or role) to add, list, or remove a tag for an instance profile

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace *<InstanceProfileName>* with the name of the instance profile whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListInstanceProfileTags",
        "iam:TagInstanceProfile",
        "iam:UntagInstanceProfile"
    ],
    "Resource": "arn:aws:iam::<account-number>:instance-profile/<InstanceProfileName>"
}
```

To allow an IAM entity (user or role) to add a tag to a specific instance profile

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific instance profile.

> **Note**
>
> The `iam:TagInstanceProfile` action requires that you also include the `iam:ListInstanceProfileTags` action.

To use this policy, replace *<InstanceProfileName>* with the name of the instance profile whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListInstanceProfileTags",
        "iam:TagInstanceProfile"
    ],
    "Resource": "arn:aws:iam::<account-number>:instance-profile/<InstanceProfileName>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on instance profiles (AWS CLI or AWS API)

You can list, attach, or remove tags for instance profiles. You can use the AWS CLI or the AWS API to manage tags for instance profiles.

To list the tags currently attached to an instance profile (AWS CLI or AWS API)

- AWS CLI: aws iam list-instance-profile-tags
- AWS API: ListInstanceProfileTags

To attach tags to an instance profile (AWS CLI or AWS API)

- AWS CLI: aws iam tag-instance-profile
- AWS API: TagInstanceProfile

To remove tags from an instance profile (AWS CLI or AWS API)

- AWS CLI: aws iam untag-instance-profile
- AWS API: UntagInstanceProfile

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag server certificates

If you use IAM to manage SSL/TLS certificates, you can tag server certificates in IAM using the AWS CLI or AWS API. For certificates in a Region supported by AWS Certificate Manager (ACM), we recommend that you use ACM instead of IAM to provision, manage, and deploy your server certificates. In unsupported Regions, you must use IAM as a certificate manager. To learn which Regions ACM supports, see AWS Certificate Manager endpoints and quotas in the *AWS General Reference*.

You can use IAM tag key-value pairs to add custom attributes to a server certificate. For example, to add information about the owner or administrator of a server certificate, add the tag key **`owner`** and the tag value **`net-eng`**. Or you can specify a cost center by adding the tag key **`CostCenter`** and the tag value **`1234`**. You can use tags to control access to resources or to control what tags can be attached to resources. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see Pass session tags in AWS STS.

Permissions required for tagging server certificates

You must configure permissions to allow an IAM entity (user or role) to tag server certificates. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListServerCertificateTags`
- `iam:TagServerCertificate`
- `iam:UntagServerCertificate`

To allow an IAM entity (user or role) to add, list, or remove a tag for a server certificate

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace *<CertificateName>* with the name of the server certificate whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListServerCertificateTags",
        "iam:TagServerCertificate",
        "iam:UntagServerCertificate"
    ],
    "Resource": "arn:aws:iam::<account-number>:server-certificate/<CertificateName>"
}
```

To allow an IAM entity (user or role) to add a tag to a specific server certificate

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific server certificate.

> **Note**
>
> The `iam:TagServerCertificate` action requires that you also include the `iam:ListServerCertificateTags` action.

To use this policy, replace *<CertificateName>* with the name of the server certificate whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListServerCertificateTags",
        "iam:TagServerCertificate"
    ],
    "Resource": "arn:aws:iam::<account-number>:server-certificate/<CertificateName>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on server certificates (AWS CLI or AWS API)

You can list, attach, or remove tags for server certificates. You can use the AWS CLI or the AWS API to manage tags for server certificates.

To list the tags currently attached to a server certificate (AWS CLI or AWS API)

- AWS CLI: aws iam list-server-certificate-tags
- AWS API: ListServerCertificateTags

To attach tags to a server certificate(AWS CLI or AWS API)

- AWS CLI: aws iam tag-server-certificate
- AWS API: TagServerCertificate

To remove tags from a server certificate (AWS CLI or AWS API)

- AWS CLI: aws iam untag-server-certificate
- AWS API: UntagServerCertificate

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag virtual MFA devices

You can use IAM tag key-value pairs to add custom attributes to a virtual MFA device. For example, to add cost center information for a user's virtual MFA device, you can add the tag key **`CostCenter`** and the tag value **`1234`**. You can use tags to control access to resources or to control what tags can be attached to an object. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

You can also use tags in AWS STS to add custom attributes when you assume a role or federate a user. For more information, see Pass session tags in AWS STS.

Permissions required for tagging virtual MFA devices

You must configure permissions to allow an IAM entity (user or role) to tag virtual MFA devices. You can specify one or all of the following IAM tag actions in an IAM policy:

- `iam:ListMFADeviceTags`
- `iam:TagMFADevice`
- `iam:UntagMFADevice`

To allow an IAM entity (user or role) to add, list, or remove a tag for a virtual MFA device

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and replace *<MFATokenID>* with the name of the virtual MFA device whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListMFADeviceTags",
        "iam:TagMFADevice",
        "iam:UntagMFADevice"
    ],
    "Resource": "arn:aws:iam::<account-number>:mfa/<MFATokenID>"
}
```

To allow an IAM entity (user or role) to add a tag to a specific virtual MFA device

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific MFA device.

> **Note**
>
> The `iam:TagMFADevice` action requires that you also include the `iam:ListMFADeviceTags` action.

To use this policy, replace *<MFATokenID>* with the name of the virtual MFA device whose tags need to be managed. To learn how to create a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

```
{
    "Effect": "Allow",
    "Action": [
        "iam:ListMFADeviceTags",
        "iam:TagMFADevice"
    ],
    "Resource": "arn:aws:iam::<account-number>:mfa/<MFATokenID>"
}
```

Alternatively, you can use an AWS managed policy such as IAMFullAccess to provide full access to IAM.

Managing tags on virtual MFA devices (AWS CLI or AWS API)

You can list, attach, or remove tags for a virtual MFA device. You can use the AWS CLI or the AWS API to manage tags for a virtual MFA device.

To list the tags currently attached to a virtual MFA device (AWS CLI or AWS API)

- AWS CLI: aws iam list-mfa-device-tags
- AWS API: ListMFADeviceTags

To attach tags to a virtual MFA device (AWS CLI or AWS API)

- AWS CLI: aws iam tag-mfa-device
- AWS API: TagMFADevice

To remove tags from a virtual MFA device (AWS CLI or AWS API)

- AWS CLI: aws iam untag-mfa-device
- AWS API: UntagMFADevice

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see IAM policy elements: Variables and tags.

Tag account access manager instances

You must configure permissions to allow an IAM entity (roles or users) to tag account access manager instances. You can specify one or all of the following account-access tag actions in an IAM policy:

- `account-access:TagResource`
- `account-access:ListTagsForResource`
- `account-access:UntagResource`

To allow an IAM entity (role or user) to add, list, or remove a tag for an account access manager instance

Add the following statement to the permissions policy for the IAM entity that needs to manage tags. Use your account number and instance IDs of your IAM Identity Center and account access manager instances to replace the placeholders. To learn how to create a policy using this example JSON policy document, see the section called “Create IAM policies (console)”.

```
{
    "Effect": "Allow",
    "Action": [
        "account-access:TagResource",
        "account-access:ListTagsForResource",
        "account-access:UntagResource"
    ],
    "Resource": "<account_access_manager_ARN>"
}
```

To allow an IAM entity (role or user) to add a tag to a specific account access manager instance

Add the following statement to the permissions policy for the IAM entity that needs to add, but not remove, tags for a specific account access manager instance.

> **Note**
>
> The `account-access:TagResource` action requires that you also include the `account-access:ListTagsForResource` action.

Use your account number and instance IDs of your IAM Identity Center and account access manager instances to replace the placeholders. To learn how to create a policy using this example JSON policy document, see the section called “Create IAM policies (console)”.

```
{
    "Effect": "Allow",
    "Action": [
        "account-access:TagResource",
        "account-access:ListTagsForResource"
    ],
    "Resource": "<account_access_manager_ARN>"
}
```

Managing tags on account access manager (console)

You can manage tags for an account access manager instance from the AWS Management Console.

To manage tags on account access manager (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Account access manager**, and then choose the **Settings** tab.
3. In the **Tags** section, you can see currently configured tags. Choose **Manage tags** and then complete one of the following actions:
   - Choose **Add new tag** to add a new tag.
   - Edit existing tag keys and values.
   - Choose **Remove** to remove a tag.
4. Add or remove tags to complete the set of tags. Then choose **Save changes**.

Managing tags on account access manager (AWS CLI or AWS API)

You can list, attach, or remove tags for account access manager instances using the AWS CLI or AWS API.

To list the tags currently attached to an account access manager instance (AWS CLI or AWS API)

- AWS CLI: `aws account-access list-tags-for-resource`
- AWS API: `account-access:ListTagsForResource`

To attach tags to an account access manager instance (AWS CLI or AWS API)

- AWS CLI: `aws account-access tag-resource`
- AWS API: `TagResource`

To remove tags from an account access manager instance (AWS CLI or AWS API)

- AWS CLI: `aws account-access untag-resource`
- AWS API: `UntagResource`

For information about attaching tags to resources for other AWS services, see the documentation for those services.

For information about using tags to set more granular permissions with IAM permissions policies, see the section called “Variables and tags”.

Pass session tags in AWS STS

Session tags are key-value pair attributes that you pass when you assume an IAM role or federate a user in AWS STS. You do this by making an AWS CLI or AWS API request through AWS STS or through your identity provider (IdP). When you use AWS STS to request temporary security credentials, you generate a session. Sessions expire and have credentials, such as an access key pair and a session token. When you use the session credentials to make a subsequent request, the request context includes the `aws:PrincipalTag` context key. You can use the `aws:PrincipalTag` key in the `Condition` element of your policies to allow or deny access based on those tags.

When you use temporary credentials to make a request, your principal might include a set of tags. These tags come from the following sources:

1. **Session tags** – The tags passed when you assume the role or federate the user using the AWS CLI or AWS API. For more information about these operations, see Session tagging operations.
2. **Incoming transitive session tags** – The tags inherited from a previous session in a role chain. For more information, see Chaining roles with session tags later in this topic.
3. **IAM tags** – The tags attached to your IAM assumed role.

Topics

- Session tagging operations
- Things to know about session tags
- Permissions required to add session tags
- Passing session tags using AssumeRole
- Passing session tags using AssumeRoleWithSAML
- Passing session tags using AssumeRoleWithWebIdentity
- Passing session tags using GetFederationToken
- Chaining roles with session tags
- Using session tags for ABAC
- Viewing session tags in CloudTrail

Session tagging operations

You can pass session tags using the following AWS CLI or AWS API operations in AWS STS. *The AWS Management Console **Switch Role** feature does not allow you to pass session tags.*

You can also set the session tags as transitive. Transitive tags persist during role chaining. For more information, see Chaining roles with session tags.

The following table compares methods for passing session tags.

Operation	Who can assume the role	Method to pass tags	Method to set transitive tags
assume-role CLI or AssumeRole API operation	IAM user or a session	Tags API parameter or --tags CLI option	TransitiveTagKeys API parameter or --transitive-tag-keys CLI option
assume-role-with-saml CLI or AssumeRoleWithSAML API operation	Any user authenticated using a SAML identity provider	PrincipalTag SAML attribute	TransitiveTagKeys SAML Attribute
assume-role-with-web-identity CLI or AssumeRoleWithWebIdentity API operation	Any user authenticated using an OIDC provider	PrincipalTag OIDC token	TransitiveTagKeys OIDC token
get-federation-token CLI or GetFederationToken API operation	IAM user or root user	Tags API parameter or --tags CLI option	Not supported

Operations that support session tagging can fail under the following conditions:

- You pass more than 50 session tags.
- The plaintext of your session tag keys exceeds 128 characters.
- The plaintext of your session tag values exceeds 256 characters.
- The total size of the plaintext of session policies exceeds 2048 characters.

- Session policies and session tags add to the session token size. If the token exceeds the size limit, the operation fails. The error message reports your session token size and the maximum allowed size, both in bytes. Monitor `SessionTokenUtilization` in the AWS STS API response to track your usage.

Things to know about session tags

Before you use session tags, review the following details about sessions and tags.

- When using session tags, trust policies for all roles connected to the identity provider (IdP) passing tags must have the `sts:TagSession` permission. For roles without this permission in the trust policy, the `AssumeRole` operation fails.
- When you request a session, you can specify principal tags as the session tags. The tags apply to requests that you make using the session's credentials.
- Session tags use key-value pairs. For example, to add contact information to a session, you can add the session tag key `email` and the tag value `johndoe@example.com`.
- Session tags must follow the rules for naming tags in IAM and AWS STS. This topic includes information about case sensitivity and restricted prefixes that apply to your session tags.
- New session tags override existing assumed role or federated user session tags with the same tag key, regardless of character case.
- You cannot pass session tags using the AWS Management Console.
- Session tags are valid only for the current session.
- Session tags support role chaining. By default, AWS STS does not pass tags to subsequent role sessions. However, you can set session tags as transitive. Transitive tags persist during role chaining and replace matching `ResourceTag` values after the evaluation of the role trust policy. For more information, see Chaining roles with session tags.
- You can use session tags to control access to resources or to control what tags can be passed into a subsequent session. For more information, see IAM tutorial: Use SAML session tags for ABAC.
- You can view the principal tags for your session, including the session tags, in the AWS CloudTrail logs. For more information, see Viewing session tags in CloudTrail.
- You must pass a single value for each session tag. AWS STS does not support multi-valued session tags.
- You can pass a maximum of 50 session tags. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

- Session policies and session tags add to the session token size. If the token exceeds the size limit, the AWS CLI or AWS API returns an error. The error message reports your session token size and the maximum allowed size, both in bytes. The `SessionTokenUtilization` response element shows what percentage of the size limit the token uses. The `PackedPolicySize` response element is deprecated. Use `SessionTokenUtilization` instead.

Permissions required to add session tags

In addition to the action that matches the API operation, you must have the following permissions-only action in your policy:

```
sts:TagSession
```

> **Important**
>
> When using session tags, the role trust policies for all roles connected to an identity provider (IdP) must have the `sts:TagSession` permission. The `AssumeRole` operation fails for any role connected to an IdP passing session tags without this permission. If you don't want to update the role trust policy for each role, you can use a separate IdP instance for passing session tags. Then, add the `sts:TagSession` permission to only the roles connected to the separate IdP.

You can use the `sts:TagSession` action with the following condition keys.

- `aws:PrincipalTag` – Compares the tag attached to the principal making the request with the tag you specified in the policy. For example, you can allow a principal to pass session tags only if the principal making the request has the specified tags.
- `aws:RequestTag` – Compares the tag key-value pair passed in the request with the tag pair you specified in the policy. For example, you can allow the principal to pass the specified session tags, but only with the specified values.
- `aws:ResourceTag` – Compares the tag key-value pair you specified in the policy with the key-value pair attached to the resource. For example, you can allow the principal to pass session tags only if the role they assume includes the specified tags.

- `aws:TagKeys` – Compares the tag keys in a request with the keys you specified in the policy. For example, you can allow the principal to pass only session tags with the specified tag keys. This condition key limits the maximum set of session tags that can be passed.
- `sts:TransitiveTagKeys` - Compares the transitive session tag keys in the request with those specified in the policy. For example, you can write a policy to allow a principal to set only specific tags as transitive. Transitive tags persist during role chaining. For more information, see Chaining roles with session tags.

For example, the following role trust policy allows the `test-session-tags` user to assume the role with the attached policy. When that user assumes the role, they must use the AWS CLI or AWS API to pass the three required session tags and the required external ID. Additionally, the user can choose to set the `Project` and `Department` tags as transitive.

Example role trust policy for session tags

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowIamUserAssumeRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/test-session-
tags"},
            "Condition": {
              "StringLike": {
                    "aws:RequestTag/Project": "*",
                    "aws:RequestTag/CostCenter": "*",
                    "aws:RequestTag/Department": "*"
                },
                "StringEquals": {"sts:ExternalId": "Example987"}
            }
        },
        {
            "Sid": "AllowPassSessionTagsAndTransitive",
            "Effect": "Allow",
            "Action": "sts:TagSession",
```

```
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/test-session-
tags"},
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/Project": "*",
                    "aws:RequestTag/CostCenter": "*"
                },
                "StringEquals": {
                    "aws:RequestTag/Department": [
                        "Engineering",
                        "Marketing"
                    ]
                },
                "ForAllValues:StringEquals": {
                    "sts:TransitiveTagKeys": [
                        "Project",
                        "Department"
                    ]
                }
            }
        }
    ]
}
```

What does this policy do?

- The `AllowIamUserAssumeRole` statement allows the `test-session-tags` user to assume the role with the attached policy. When that user assumes the role, they must pass the required session tags and external ID.
  - The first condition block of this statement requires the user to pass the `Project`, `CostCenter`, and `Department` session tags. The tag values do not matter in this statement, so you can use wildcards (*) for the tag values. This block ensures that user passes at least these three session tags. Otherwise, the operation fails. The user can pass additional tags.
  - The second condition block requires the user to pass an external ID with the value `Example987`.
- The `AllowPassSessionTagsAndTransitive` statement allows the `sts:TagSession` permissions-only action. This action must be allowed before the user can pass session tags. If your policy includes the first statement without the second statement, the user can't assume the role.

- The first condition block of this statement allows the user to pass any value for the `CostCenter` and `Project` session tags. You do this by using wildcards (*) for the tag value in the policy, which requires that you use the StringLike condition operator.
- The second condition block allows the user to pass only the `Engineering` or `Marketing` value for the `Department` session tag.
- The third condition block lists the maximum set of tags you can set as transitive. The user can choose to set a subset or no tags as transitive. They cannot set additional tags as transitive. You can require that they set at least one of the tags as transitive by adding another condition block that includes `"Null":{"sts:TransitiveTagKeys":"false"}`.

Passing session tags using AssumeRole

The `AssumeRole` operation returns a set of temporary credentials you can use to access AWS resources. You can use IAM user or role credentials to call `AssumeRole`. To pass session tags while assuming a role, use the `--tags` AWS CLI option or the `Tags` AWS API parameter.

To set tags as transitive, use the `--transitive-tag-keys` AWS CLI option or the `TransitiveTagKeys` AWS API parameter. Transitive tags persist during role chaining. For more information, see Chaining roles with session tags.

The following example shows a sample request that uses `AssumeRole`. In this example, when you assume the `my-role-example` role, you create a session named `my-session`. You add the session tag key-value pairs `Project` = `Automation`, `CostCenter` = `12345`, and `Department` = `Engineering`. You also set the `Project` and `Department` tags as transitive by specifying their keys. You must pass a single value for each session tag. AWS STS does not support multi-valued session tags.

Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/my-role-example \
--role-session-name my-session \
--tags Key=Project,Value=Automation Key=CostCenter,Value=12345
 Key=Department,Value=Engineering \
--transitive-tag-keys Project Department \
--external-id Example987
```

Passing session tags using AssumeRoleWithSAML

The `AssumeRoleWithSAML` operation authenticates with SAML-based federation. This operation returns a set of temporary credentials you can use to access AWS resources. For more information about using SAML-based federation for AWS Management Console access, see Enabling SAML 2.0 federated principals to access the AWS Management Console. For details about AWS CLI or AWS API access, see SAML 2.0 federation. For a tutorial on configuring SAML federation for your Active Directory users, see AWS Federated Authentication with Active Directory Federation Services (ADFS) in the AWS Security Blog.

As an administrator, you can allow members of your company directory to federate into AWS using the AWS STS `AssumeRoleWithSAML` operation. To do this, you must complete the following tasks:

1. Configure your network as a SAML provider for AWS.
2. Create a SAML provider in IAM
3. Create a role for SAML 2.0 federation (console)
4. Finish configuring the SAML IdP and create assertions for the SAML authentication response

AWS includes identity providers with certified end-to-end experience for session tags with their identity solutions. To learn how to use these identity providers to configure session tags, see Integrate third-party SAML solution providers with AWS.

To pass SAML attributes as session tags, include the `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/PrincipalTag:`*`{TagKey}`*. Use the `AttributeValue` element to specify the value of the tag. Include a separate `Attribute` element for each session tag.

For example, assume that you want to pass the following identity attributes as session tags:

- `Project:Automation`
- `CostCenter:12345`
- `Department:Engineering`

To pass these attributes, include the following elements in your SAML assertion.

Example snippet of a SAML assertion

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:Project">
```

```
  <AttributeValue>Automation</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:CostCenter">
  <AttributeValue>12345</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:Department">
  <AttributeValue>Engineering</AttributeValue>
</Attribute>
```

To set the preceding tags as transitive, include another `Attribute` element with the `Name` attribute set to `https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys`. Transitive tags persist during role chaining. For more information, see Chaining roles with session tags.

To set the `Project` and `Department` tags as transitive, use the following multi-valued attribute:

Example snippet of a SAML assertion

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/TransitiveTagKeys">
  <AttributeValue>Project</AttributeValue>
  <AttributeValue>Department</AttributeValue>
</Attribute>
```

Passing session tags using AssumeRoleWithWebIdentity

Use OpenID Connect (OIDC)-compliant federation to authenticate the `AssumeRoleWithWebIdentity` operation. This operation returns a set of temporary credentials you can use to access AWS resources. For more information about using web identity federation for AWS Management Console access, see OIDC federation.

To pass session tags from OpenID Connect (OIDC), you must include the session tags in the JSON Web Token (JWT) when you submit the `AssumeRoleWithWebIdentity` request. To learn more about OIDC tokens and claims, see Using Tokens with User Pools in the *Amazon Cognito Developer Guide*.

AWS supports two claim formats for including session tags in the JWT:

- Nested claim format
- Flattened claim format

Nested claim format

The nested claim format uses a structure within the `https://aws.amazon.com/tags` namespace in the JWT. In this format:

- Principal tags are represented as a nested object under the `principal_tags` key.
- Each principal tag is a single string value.
- Transitive tag keys are represented in an array under the `transitive_tag_keys` key.
- Both `principal_tags` and `transitive_tag_keys` are nested under the `https://aws.amazon.com/tags` namespace.

The following example demonstrates a decoded JWT using the nested object format:

Example decoded JSON Web Token using the nested claim format

```
{
    "sub": "johndoe",
    "aud": "ac_oic_client",
    "jti": "ZYUCeRMQVtqHypVPWAN3VB",
    "iss": "https://xyz.com",
    "iat": 1566583294,
    "exp": 1566583354,
    "auth_time": 1566583292,
    "https://aws.amazon.com/tags": {
        "principal_tags": {
            "Project": ["Automation"],
            "CostCenter": ["987654"],
            "Department": ["Engineering"]
        },
        "transitive_tag_keys": [
            "Project",
            "CostCenter"
        ]
    }
}
```

Flattened claim format

The flattened claim format is compatible with identity providers that don't support nested objects in JWT claims, such as Microsoft Entra ID. In this format:

- Principal tags are represented as separate claims with the prefix `https://aws.amazon.com/tags/principal_tags/`.
- Each principal tag is a single string value.
- Transitive tag keys are represented in a single claim as an array of strings with the prefix `https://aws.amazon.com/tags/transitive_tag_keys`.

Now, let's look at how the same information is represented using the flattened claim format:

Example decoded JSON Web Token using the flattened claim format

```
{
    "sub": "johndoe",
    "aud": "ac_oic_client",
    "jti": "ZYUCeRMQVtqHypVPWAN3VB",
    "iss": "https://xyz.com",
    "iat": 1566583294,
    "exp": 1566583354,
    "auth_time": 1566583292,
    "https://aws.amazon.com/tags/principal_tags/Project": "Automation",
    "https://aws.amazon.com/tags/principal_tags/CostCenter": "987654",
    "https://aws.amazon.com/tags/principal_tags/Department": "Engineering",
    "https://aws.amazon.com/tags/transitive_tag_keys": [
        "Project",
        "CostCenter"
    ]
}
```

Both decoded JWT examples show a call to `AssumeRoleWithWebIdentity` with the `Project`, `CostCenter`, and `Department` session tags. Both tokens set the `Project` and `CostCenter` tags as transitive. Transitive tags persist during role chaining. For more information, see Chaining roles with session tags.

The flattened claim format achieves the same result as the nested claim format but uses a flattened structure for tags. It allows you to include session tags in environments where nested JSON objects are not supported in JWT claims. When using either format, ensure that your identity provider is configured to issue tokens with the appropriate claim structures. AWS supports both claim formats, so you can choose the one that best fits your identity provider's specific requirements.

Passing session tags using GetFederationToken

The `GetFederationToken` allows you to federate your user. This operation returns a set of temporary credentials you can use to access AWS resources. To add tags to your federated user session, use the `--tags` AWS CLI option or the `Tags` AWS API parameter. You can't set session tags as transitive when you use `GetFederationToken`, because you can't use the temporary credentials to assume a role. You cannot use role chaining in this case.

The following example shows a sample request using `GetFederationToken`. In this example, when you request the token, you create a session named `my-fed-user`. You add the session tag key-value pairs `Project` = `Automation` and `Department` = `Engineering`.

Example GetFederationToken CLI request

```
aws sts get-federation-token \
--name my-fed-user \
--tags key=Project,value=Automation key=Department,value=Engineering
```

When you use the temporary credentials returned by the `GetFederationToken` operation, the session principal tags include the user tags and the passed session tags.

Chaining roles with session tags

You can assume one role and then use the temporary credentials to assume another role. You can continue from session to session. This is called role chaining. When you pass session tags while assuming a role, you can set the keys as transitive. This ensures that those session tags pass to subsequent sessions in a role chain. You cannot set role tags as transitive. To pass these tags to subsequent sessions, specify them as session tags.

> **Note**
>
> Transitive tags persist during role chaining and replace matching `ResourceTag` values after the evaluation of the role trust policy.

The following example shows how AWS STS passes session tags, transitive tags, and role tags into subsequent sessions in a role chain.

In this example role chaining scenario, you use an IAM user access key in the AWS CLI to assume a role named `Role1`. You then use the resulting session credentials to assume a second role named

Role2. You can then use the second session credentials to assume a third role named Role3. These requests occur as three separate operations. Each role is already tagged in IAM. And during each request, you pass additional session tags.
Role chaining.

When you chain roles, you can ensure that tags from an earlier session persist to the later sessions. To do this using the assume-role CLI command, you must pass the tag as a session tag and set the tag as transitive. You pass the tag Star = 1 as a session tag. The command also attaches the tag Heart = 1 to the role and applies as a principal tag when you use the session. However, you also want the Heart = 1 tag to automatically pass to the second or third session. To do that, you manually include it as a session tag. The resulting session principal tags include both of these tags, and sets them as transitive.
Assuming the first role in a role chain.

You perform this request using the following AWS CLI command:

Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/Role1 \
--role-session-name Session1 \
--tags Key=Star,Value=1 Key=Heart,Value=1 \
--transitive-tag-keys Star Heart
```

You then use the credentials for that session to assume Role2. The command attaches the tag Sun = 2 to the second role and applies as a principal tag when you use the second session. The Heart and Star tags inherit the transitive session tags in the first session. The second session resulting principal tags are Heart = 1, Star = 1, and Sun = 2. Heart and Star continue to be transitive. The Sun tag attached to Role2 is not marked as transitive because it is not a session tag. Future sessions do not inherit this tag.
Assuming the second role in a role chain.

You perform this second request using the following AWS CLI command:

Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/Role2 \
--role-session-name Session2
```

You then use the second session credentials to assume `Role3`. The principal tags for the third session come from any new session tags, the inherited transitive session tags, and the role tags. The `Heart` = `1` and `Star` = `1` tags on the second session are inherited from the transitive session tag in the first session. If you try to pass the Sun = 2 session tag, the operation fails. The inherited `Star` = `1` session tag overrides the role `Star` = `3` tag. In role chaining, the value of a transitive tag overrides the role matching the `ResourceTag` value after the evaluation of the role trust policy. In this example, if `Role3` uses `Star` as a `ResourceTag` in the role trust policy, and sets `ResourceTag` value to the transitive tag value from the calling role session. The role `Lightning` tag also applies to the third session, and is not set as transitive.
Assuming the third role in a role chain.

You perform the third request using the following AWS CLI command:

Example AssumeRole CLI request

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/Role3 \
--role-session-name Session3
```

Using session tags for ABAC

Attribute-based access control (ABAC) uses an authorization strategy that defines permissions based on tag attributes.

If your company uses an OIDC or SAML-based identity provider (IdP) to manage user identities, you can configure your assertion to pass session tags to AWS. For example, with corporate user identities, when your employees federate into AWS, AWS applies their attributes to their resulting principal. You can then use ABAC to allow or deny permissions based on those attributes. For details, see IAM tutorial: Use SAML session tags for ABAC.

For more information about using IAM Identity Center with ABAC, see Attributes for access control in the *AWS IAM Identity Center User Guide*.

Viewing session tags in CloudTrail

You can use AWS CloudTrail to view the requests used to assume roles or federate users. The CloudTrail log file includes information about the principal tags for the assumed-role or federated user session. For more information, see Logging IAM and AWS STS API calls with AWS CloudTrail.

For example, assume that you make an AWS STS `AssumeRoleWithSAML` request, pass session tags, and set those tags as transitive. You can find the following information in your CloudTrail log.

Example AssumeRoleWithSAML CloudTrail log

```
    "requestParameters": {
        "sAMLAssertionID": "_c0046cEXAMPLEb9d4b8eEXAMPLE2619aEXAMPLE",
        "roleSessionName": "MyRoleSessionName",
        "principalTags": {
            "CostCenter": "987654",
            "Project": "Unicorn"
        },
        "transitiveTagKeys": [
            "CostCenter",
            "Project"
        ],
        "durationSeconds": 3600,
        "roleArn": "arn:aws:iam::123456789012:role/SAMLTestRoleShibboleth",
        "principalArn": "arn:aws:iam::123456789012:saml-provider/Shibboleth"
    },
```

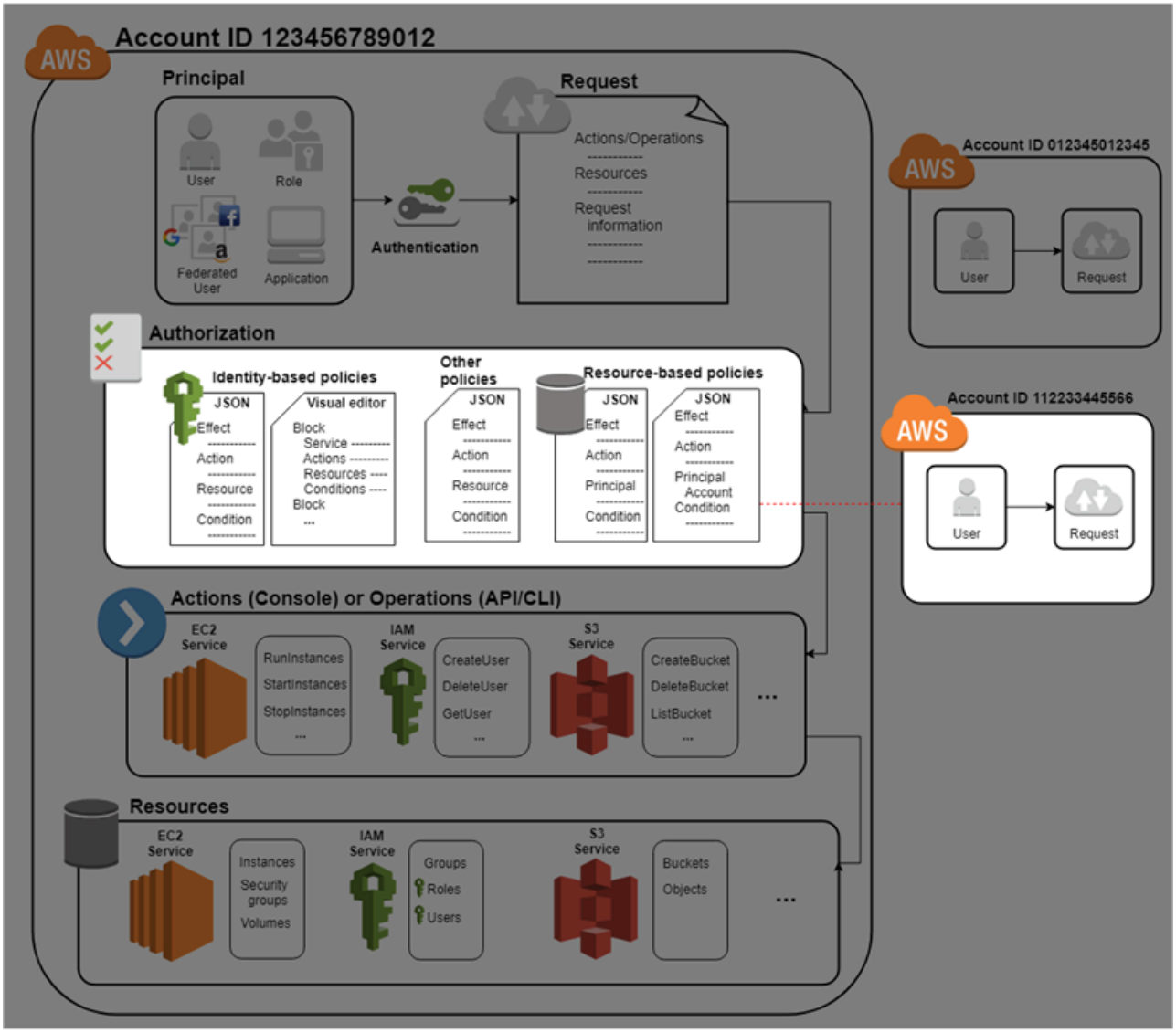
You can view the following example CloudTrail logs to view events that use session tags.

- Example AWS STS role chaining API event in CloudTrail log file
- Example SAML AWS STS API event in CloudTrail log file
- Example OIDC AWS STS API event in CloudTrail log file

Access management for AWS resources

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. When a principal makes a request in AWS, the AWS enforcement code checks whether the principal is authenticated (signed in) and authorized (has permissions). You manage access in AWS by creating policies and attaching them to IAM identities or AWS resources. Policies are JSON documents in AWS that, when attached to an identity or resource, define their permissions. For more information about policy types and uses, see Policies and permissions in AWS Identity and Access Management.

For details about the rest of the authentication and authorization process, see How IAM works.



During authorization, the AWS enforcement code uses values from the request context to check for matching policies and determine whether to allow or deny the request.

AWS checks each policy that applies to the context of the request. If a single policy denies the request, AWS denies the entire request and stops evaluating policies. This is called an *explicit deny*. Because requests are *denied by default*, IAM authorizes your request only if every part of your request is allowed by the applicable policies. The evaluation logic for a request within a single account follows these rules:

- By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access.)
- An explicit allow in an identity-based or resource-based policy overrides this default.
- If a permissions boundary, AWS Organizations SCP, or session policy is present, it might override the allow with an implicit deny.
- An explicit deny in any policy overrides any allows.

After your request has been authenticated and authorized, AWS approves the request. If you need to make a request in a different account, a policy in the other account must allow you to access the resource. In addition, the IAM entity that you use to make the request must have an identity-based policy that allows the request.

You can also use IAM temporary delegation to grant Amazon and AWS Partners limited, time-bounded access to configure AWS services in your account.

Access management resources

For more information about permissions and about creating policies, see the following resources:

The following entries in the AWS Security Blog cover common ways to write policies for access to Amazon S3 buckets and objects.

- Writing IAM Policies: How to Grant Access to an Amazon S3 Bucket
- Writing IAM policies: Grant Access to User-Specific Folders in an Amazon S3 Bucket
- IAM Policies and Bucket Policies and ACLs! Oh My! (Controlling Access to S3 Resources)
- A Primer on RDS Resource-Level Permissions
- Demystifying EC2 Resource-Level Permissions

Policies and permissions in AWS Identity and Access Management

Manage access in AWS by creating policies and attaching them to IAM identities (users, groups of users, or roles) or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an IAM principal (user or role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. AWS supports nine types of policies: identity-based policies, resource-based policies, VPC endpoint policies, permissions boundaries, AWS Organizations service control policies (SCPs), AWS Organizations resource control policies (RCPs), access control lists (ACLs), Resource Access Manager shares (RAM) and session policies.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, if a policy allows the GetUser action, then a user with that policy can get user information from the AWS Management Console, the AWS CLI, or the AWS API. When you create an IAM user, you can choose to allow console or programmatic access. If console access is allowed, the IAM user can sign in to the console using their sign-in credentials. If programmatic access is allowed, the user can use access keys to work with the CLI or API.

Policy types

The following policy types, listed in order from most frequently used to less frequently used, are available for use in AWS. For more details, see the sections below for each policy type.

- **Identity-based policies** – Attach managed and inline policies to IAM identities (users, groups to which users belong, or roles). Identity-based policies grant permissions to an identity.
- **Resource-based policies** – Attach inline policies to resources. The most common examples of resource-based policies are Amazon S3 bucket policies and IAM role trust policies. Resource-based policies grant permissions to the principal that is specified in the policy. Principals can be in the same account as the resource or in other accounts.
- **VPC endpoint policies** – Attach to a VPC endpoint to control which principals can use the VPC endpoint policies and which resources can be accessed through it. VPC endpoint policies act as an additional access boundary scoped to traffic that traverses the endpoint.
- **Permissions boundaries** – Use a managed policy as the permissions boundary for an IAM entity (user or role). That policy defines the maximum permissions that the identity-based policies can

grant to an entity, but does not grant permissions. Permissions boundaries do not define the maximum permissions that a resource-based policy can grant to an entity.

- **AWS Organizations SCPs** – Use an AWS Organizations service control policy (SCP) to define the maximum permissions for IAM users and IAM roles within accounts in your organization or organizational unit (OU). SCPs limit permissions that identity-based policies or resource-based policies grant to IAM users or IAM roles within the account. SCPs do not grant permissions.
- **AWS Organizations RCPs** – Use an AWS Organizations resource control policy (RCP) to define the maximum permissions for resources within accounts in your organization or organizational unit (OU). RCPs limit permissions that identity-based and resource-based policies can grant to resources in accounts within your organization. RCPs do not grant permissions.
- **Access control lists (ACLs)** – Use ACLs to control which principals in other accounts can access the resource to which the ACL is attached. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document structure. ACLs are cross-account permissions policies that grant permissions to the specified principal. ACLs cannot grant permissions to entities within the same account.
- **AWS RAM resource shares** – Use AWS Resource Access Manager to share resources across AWS accounts, organizational units, or an entire organization without writing individual resource-based policies for each shared resource.
- **Session policies** – Pass advanced session policies when you use the AWS CLI or AWS API to assume a role or a federated user. Session policies limit the permissions that the role or user's identity-based policies grant to the session. Session policies limit permissions for a created session, but do not grant permissions. For more information, see Session Policies.

Identity-based policies

Identity-based policies are JSON permissions policy documents that control what actions an identity (users, groups of users, and roles) can perform, on which resources, and under what conditions. Identity-based policies can be further categorized:

- **Managed policies** – Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. There are two types of managed policies:
  - **AWS managed policies** – Managed policies that are created and managed by AWS.
  - **Customer managed policies** – Managed policies that you create and manage in your AWS account. Customer managed policies provide more precise control over your policies than AWS managed policies.

- **Inline policies** – Policies that you add directly to a single user, group, or role. Inline policies maintain a strict one-to-one relationship between a policy and an identity. They are deleted when you delete the identity.

To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. These policies grant the specified principal permission to perform specific actions on that resource and defines under what conditions this applies. Resource-based policies are inline policies. There are no managed resource-based policies.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in separate AWS accounts, you must also use an identity-based policy to grant the principal access to the resource. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For step-by step instructions for granting cross-service access, see IAM tutorial: Delegate access across AWS accounts using IAM roles.

The IAM service supports only one type of resource-based policy called a role *trust policy*, which is attached to an IAM role. An IAM role is both an identity and a resource that supports resource-based policies. For that reason, you must attach both a trust policy and an identity-based policy to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. To learn how IAM roles are different from other resource-based policies, see Cross account resource access in IAM.

To see which other services support resource-based policies, see AWS services that work with IAM. To learn more about resource-based policies, see Identity-based policies and resource-based policies. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

VPC endpoint policies

A VPC endpoint policy is a resource-based policy that you attach to a VPC endpoint to control which principals can use the endpoint and which resources can be accessed through it. VPC endpoint policies do not override or replace identity-based policies or resource-based policies

attached to the destination service—they act as an additional access boundary scoped to traffic that traverses the endpoint.

VPC endpoint policies are used to ensure that only trusted identities in your AWS Organizations organization can access trusted resources through your VPC endpoints. If you do not attach a custom endpoint policy, AWS attaches a default policy that allows full access. For more information about endpoint policies, see Control access to VPC endpoints using endpoint policies in the *AWS PrivateLink Guide*.

IAM permissions boundaries

A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity. When you set a permissions boundary for an entity, the entity can perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

If you specify a role session or user in the principal element of a resource-based policy, an explicit allow in the permission boundary is not required. However, if you specify a role ARN in the principal element of a resource-based policy, an explicit allow in the permission boundary is required. In both cases, an explicit deny in the permission boundary is effective. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities.

AWS Organizations service control policies (SCPs)

If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. SCPs are JSON policies that specify the maximum permissions for IAM users and IAM roles within accounts of an organization or organizational unit (OU). The SCP limits permissions for principals in member accounts, including each AWS account root user. An explicit deny in any of these policies overrides an allow in other policies.

For more information about AWS Organizations and SCPs, see Service control policies (SCPs) in the *AWS Organizations User Guide*.

AWS Organizations resource control policies (RCPs)

If you enable all features in an organization, then you can use resource control policies (RCPs) to centrally apply access controls on resources across multiple AWS accounts. RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for

resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. An explicit deny in any applicable RCP overrides an allow in other policies that might be attached to individual identities or resources.

For more information about AWS Organizations and RCPs including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.

Access control lists (ACLs)

Access control lists (ACLs) are service policies that allow you to control which principals in another account can access a resource. ACLs cannot be used to control access for a principal within the same account. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

AWS Resource Access Manager (AWS RAM) resource shares

AWS Resource Access Manager (AWS RAM) lets you share resources you create in one AWS account with other AWS accounts, organizational units, or an entire organization in AWS Organizations, without writing individual resource-based policies for each shared resource. While you can grant cross-account access by attaching a resource-based policy directly to a resource, AWS RAM provides a managed, centralized alternative that eliminates the need to enumerate account IDs in policies or maintain identical policy documents across accounts.

With RAM, consuming accounts see shared resources natively in their service consoles, resource owners retain full ownership and visibility into who has access, and managed permissions define the maximum actions consumers can perform—all governed by a single resource share rather than per-resource policies. Common use cases include sharing VPC subnets, AWS Transit Gateway attachments, Route 53 Resolver rules, and License Manager configurations across accounts. For more information, see What is AWS Resource Access Manager? in the *AWS RAM User Guide*.

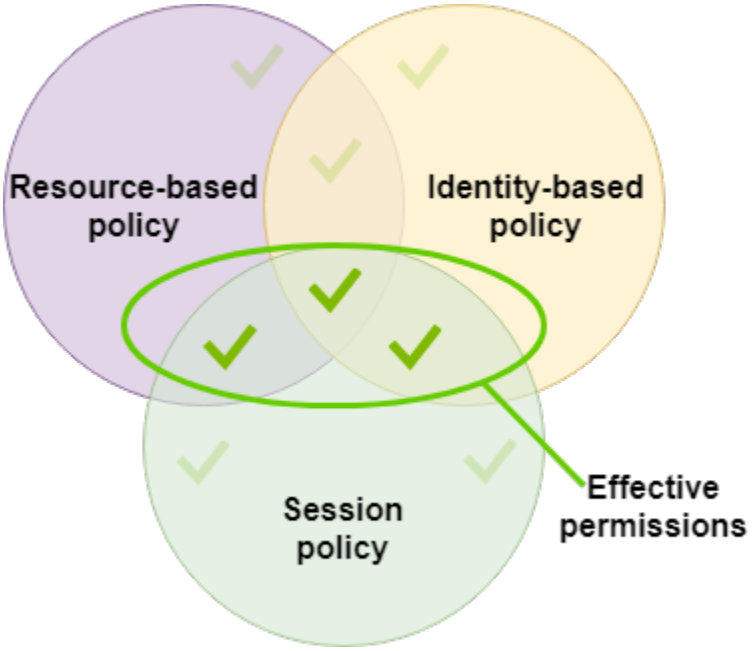
Session policies

Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or an AWS STS federated user principal. The permissions for a session are the intersection of the identity-based policies for the IAM entity (user or role) used to create the session and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow.

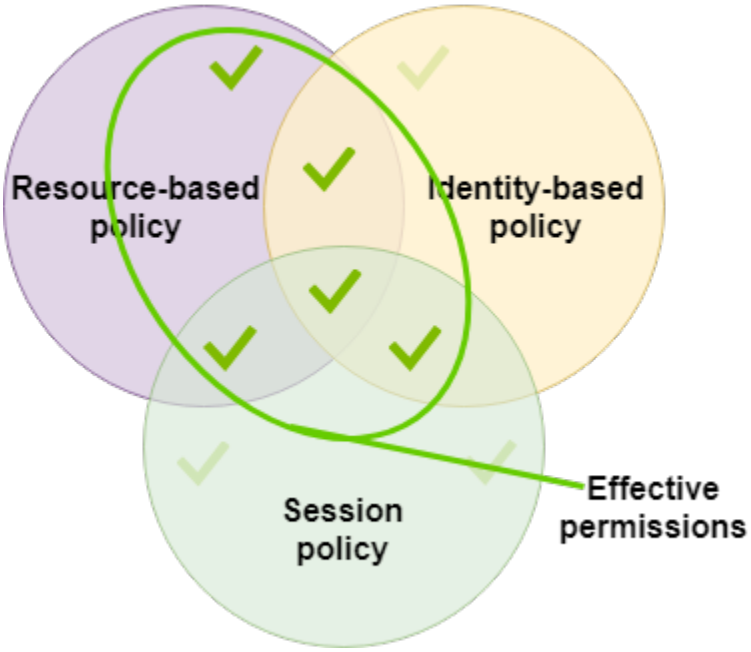
You can create role session and pass session policies programmatically using the `AssumeRole`, `AssumeRoleWithSAML`, or `AssumeRoleWithWebIdentity` API operations. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. For more information about creating a role session, see Permissions for temporary security credentials.

When you create an AWS STS federated user principal session, you use the access keys of the IAM user to programmatically call the `GetFederationToken` API operation. You must also pass session policies. The resulting session's permissions are the intersection of the identity-based policy and the session policy. For more information about creating a federated user session, see Requesting credentials through a custom identity broker.

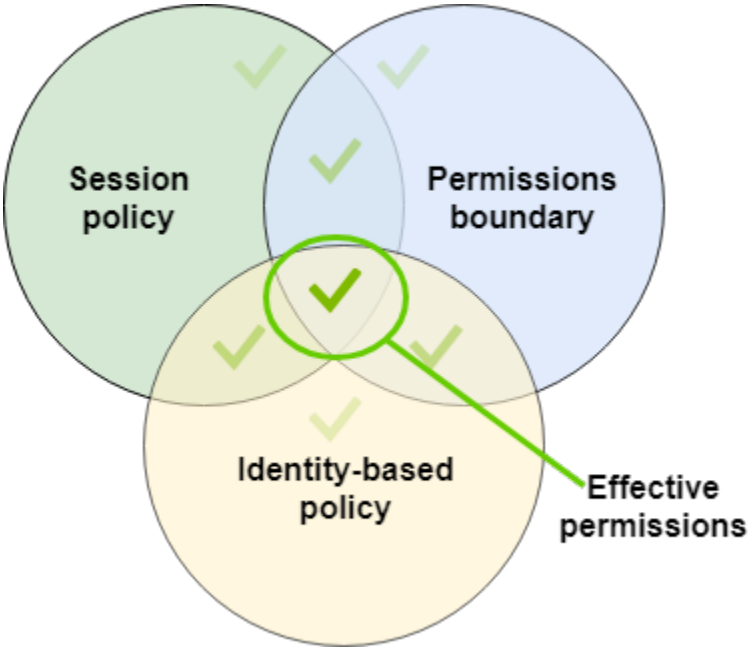
A resource-based policy can specify the ARN of the user or role as a principal. In that case, the permissions from the resource-based policy are added to the role or user's identity-based policy before the session is created. The session policy limits the total permissions granted by the resource-based policy and the identity-based policy. The resulting session's permissions are the intersection of the session policies and the resource-based policies plus the intersection of the session policies and identity-based policies.



A resource-based policy can specify the ARN of the session as a principal. In that case, the permissions from the resource-based policy are added after the session is created. The resource-based policy permissions are not limited by the session policy. The resulting session has all the permissions of the resource-based policy *plus* the intersection of the identity-based policy and the session policy.



A permissions boundary can set the maximum permissions for a user or role that is used to create a session. In that case, the resulting session's permissions are the intersection of the session policy, the permissions boundary, and the identity-based policy. However, a permissions boundary does not limit permissions granted by a resource-based policy that specifies the ARN of the resulting session.



Policies and the root user

The AWS account root user is affected by some policy types but not others. You cannot attach identity-based policies to the root user, and you cannot set the permissions boundary for the root user. However, you can specify the root user as the principal in a resource-based policy or an ACL. A root user is still the member of an account. If that account is a member of an organization in AWS Organizations, the root user is affected by SCPs and RCPs for the account.

Overview of JSON policies

Most policies are stored in AWS as JSON documents. Identity-based policies and policies used to set permissions boundaries are JSON policy documents that you attach to a user or role. Resource-based policies are JSON policy documents that you attach to a resource. SCPs and RCPs are JSON policy documents with restricted syntax that you attach to the AWS Organizations' organization root, organizational unit (OU), or an account. ACLs are also attached to a resource, but you must use a different syntax. Session policies are JSON policies that you provide when you assume a role or federated user session.

It is not necessary for you to understand the JSON syntax. You can use the visual editor in the AWS Management Console to create and edit customer managed policies without ever using JSON. However, if you use inline policies for groups or complex policies, you must still create and edit those policies in the JSON editor using the console. For more information about using the visual editor, see Define custom IAM permissions with customer managed policies and Edit IAM policies.

When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see IAM policy validation. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation.

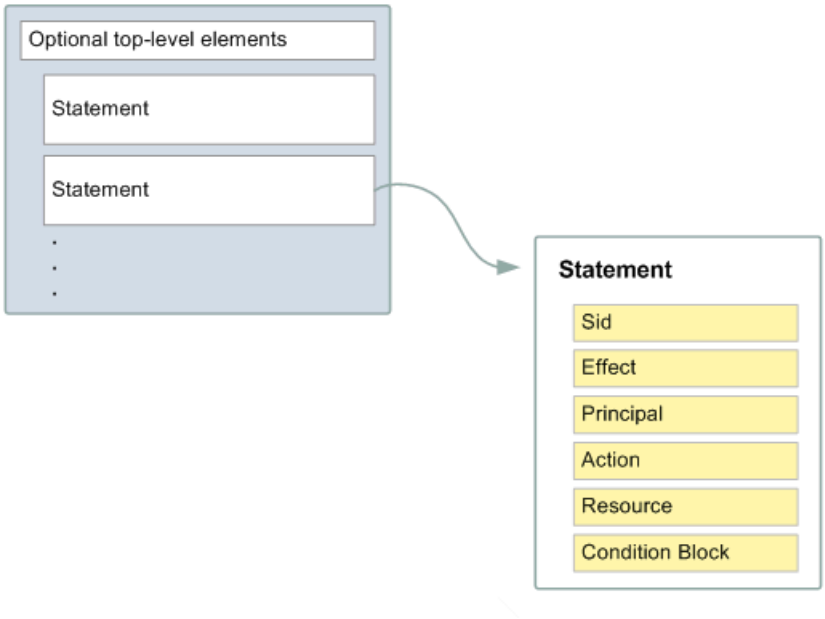
JSON policy document structure

As illustrated in the following figure, a JSON policy document includes these elements:

- Optional policy-wide information at the top of the document
- One or more individual statements

Each statement includes information about a single permission. If a policy includes multiple statements, AWS applies a logical OR across the statements when evaluating them. If multiple

policies apply to a request, AWS applies a logical OR across all of those policies when evaluating them.



The information in a statement is contained within a series of elements.

- **Version** – Specify the version of the policy language that you want to use. We recommend that you use the latest `2012-10-17` version. For more information, see IAM JSON policy elements: Version
- **Statement** – Use this main policy element as a container for the following elements. You can include more than one statement in a policy.
- **Sid** (Optional) – Include an optional statement ID to differentiate between your statements.
- **Effect** – Use `Allow` or `Deny` to indicate whether the policy allows or denies access.
- **Principal** (Required in some circumstances) – If you create a resource-based policy, you must indicate the account, user, role, or AWS STS federated user principal to which you would like to allow or deny access. If you are creating an IAM permissions policy to attach to a user or role, you cannot include this element. The principal is implied as that user or role.
- **Action** – Include a list of actions that the policy allows or denies.
- **Resource** (Required in some circumstances) – If you create an IAM permissions policy, you must specify a list of resources to which the actions apply. If you create a resource-based policy, it depends on the resource you're using as to whether this element is required or not.
- **Condition** (Optional) – Specify the circumstances under which the policy grants permission.

To learn about these and other more advanced policy elements, see IAM JSON policy element reference.

Multiple statements and multiple policies

If you want to define more than one permission for an entity (user or role), you can use multiple statements in a single policy. You can also attach multiple policies. If you try to define multiple permissions in a single statement, your policy might not grant the access that you expect. We recommend that you break up policies by resource type.

Because of the limited size of policies, it might be necessary to use multiple policies for more complex permissions. It's also a good idea to create functional groupings of permissions in a separate customer managed policy. For example, Create one policy for IAM user management, one for self-management, and another policy for S3 bucket management. Regardless of the combination of multiple statements and multiple policies, AWS evaluates your policies the same way.

For example, the following policy has three statements, each of which defines a separate set of permissions within a single account. The statements define the following:

- The first statement, with an `Sid` (Statement ID) of `FirstStatement`, lets the user with the attached policy change their own password. The `Resource` element in this statement is "`*`" (which means "all resources"). But in practice, the `ChangePassword` API operation (or equivalent `change-password` CLI command) affects only the password for the user who makes the request.
- The second statement lets the user list all the Amazon S3 buckets in their AWS account. The `Resource` element in this statement is "`*`" (which means "all resources"). But because policies don't grant access to resources in other accounts, the user can list only the buckets in their own AWS account.
- The third statement lets the user list and retrieve any object that is in a bucket named `amzn-s3-demo-bucket-confidential-data`, but only when the user is authenticated with multi-factor authentication (MFA). The `Condition` element in the policy enforces the MFA authentication.

  When a policy statement contains a `Condition` element, the statement is only in effect when the `Condition` element evaluates to true. In this case, the `Condition` evaluates to true when the user is MFA-authenticated. If the user is not MFA-authenticated, this `Condition` evaluates to false. In that case, the third statement in this policy does not apply and the user does not have access to the `amzn-s3-demo-bucket-confidential-data` bucket.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "FirstStatement",
      "Effect": "Allow",
      "Action": ["iam:ChangePassword"],
      "Resource": "*"
    },
    {
      "Sid": "SecondStatement",
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    },
    {
      "Sid": "ThirdStatement",
      "Effect": "Allow",
      "Action": [
        "s3:List*",
        "s3:Get*"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket-confidential-data",
        "arn:aws:s3:::amzn-s3-demo-bucket-confidential-data/*"
      ],
      "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
  ]
}
```

Examples of JSON policy syntax

The following identity-based policy allows the implied principal to list a single Amazon S3 bucket named `amzn-s3-demo-bucket`:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
  }
}
```

The following resource-based policy can be attached to an Amazon S3 bucket. The policy allows members of a specific AWS account to perform any Amazon S3 actions in the bucket named `amzn-s3-demo-bucket`. It allows any action that can be performed on a bucket or the objects within it. (Because the policy grants trust only to the account, individual users in the account must still be granted permissions for the specified Amazon S3 actions.)

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

To view example policies for common scenarios, see Example IAM identity-based policies.

Grant least privilege

When you create IAM policies, follow the standard security advice of granting *least privilege*, or granting only the permissions required to perform a task. Determine what users and roles need to do and then craft policies that allow them to perform *only* those tasks.

Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later.

As an alternative to least privilege, you can use AWS managed policies or policies with wildcard * permissions to get started with policies. Consider the security risk of granting your principals more permissions than they need to do their job. Monitor those principals to learn which permissions they are using. Then write least privilege policies.

IAM provides several options to help you refine the permissions that you grant.

- **Understand access level groupings** – You can use access level groupings to understand the level of access that a policy grants. Policy actions are classified as `List`, `Read`, `Write`, `Permissions management`, or `Tagging`. For example, you can choose actions from the `List` and `Read` access levels to grant read-only access to your users. To learn how to use policy summaries to understand access level permissions, see Access levels in policy summaries.
- **Validate your policies** – You can perform policy validation using IAM Access Analyzer when you create and edit JSON policies. We recommend that you review and validate all of your existing policies. IAM Access Analyzer provides over 100 policy checks to validate your policies. It generates security warnings when a statement in your policy allows access we consider overly permissive. You can use the actionable recommendations that are provided through the security warnings as you work toward granting least privilege. To learn more about policy checks provided by IAM Access Analyzer, see IAM Access Analyzer policy validation.
- **Generate a policy based on access activity** – To help you refine the permissions that you grant, you can generate an IAM policy that is based on the access activity for an IAM entity (user or role). IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that have been used by the entity in your specified time frame. You can use the template to create a managed policy with fine-grained permissions and then attach it to the IAM entity. That way, you grant only the permissions that the user or role needs

to interact with AWS resources for your specific use case. To learn more, see IAM Access Analyzer policy generation.

- **Use last accessed information** – Another feature that can help with least privilege is *last accessed information*. View this information on the **Access Advisor** tab on the IAM console details page for an IAM user, group, role, or policy. Last accessed information also includes information about the actions that were last accessed for some services, such as Amazon EC2, IAM, Lambda, and Amazon S3. If you sign in using AWS Organizations management account credentials, you can view service last accessed information in the **AWS Organizations** section of the IAM console. You can also use the AWS CLI or AWS API to retrieve a report for last accessed information for entities or policies in IAM or AWS Organizations. You can use this information to identify unnecessary permissions so that you can refine your IAM or AWS Organizations policies to better adhere to the principle of least privilege. For more information, see Refine permissions in AWS using last accessed information.
- **Review account events in AWS CloudTrail** – To further reduce permissions, you can view your account's events in AWS CloudTrail **Event history**. CloudTrail event logs include detailed event information that you can use to reduce the policy's permissions. The logs include only the actions and resources that your IAM entities need. For more information, see Viewing CloudTrail Events in the CloudTrail Console in the *AWS CloudTrail User Guide*.

For more information, see the following policy topics for individual services, which provide examples of how to write policies for service-specific resources.

- Using resource-based policies for DynamoDB in the *Amazon DynamoDB Developer Guide*
- Bucket policies for Amazon S3 in the *Amazon Simple Storage Service User Guide*
- Access Control List (ACL) overview in the *Amazon Simple Storage Service User Guide*

Managed policies and inline policies

When you set the permissions for an identity in IAM, you must decide whether to use an AWS managed policy, a customer managed policy, or an inline policy. The following topics provide more information about each of the types of identity-based policies and when to use them.

The following table outlines these policies:

Policy Type	Description	Who manages the policy?	Modify permissions?	Number of principals applied to policy?
AWS managed policies	Standalone policy created and administe red by AWS.	AWS	No	Many
Customer managed policies	Policy you create for specific use cases, and you can change or update them as often as you like.	You	Yes	Many
Inline policies	Policy created for a single IAM identity (user, group, or role) that maintains a strict one-to-one relationship between a policy and an identity.	You	Yes	One

For instructions on creating roles and attaching these policies, see Creating roles and attaching policies (console).

Topics

- AWS managed policies
- Customer managed policies
- Inline policies
- Choose between managed policies and inline policies

- Convert an inline policy to a managed policy
- Deprecated AWS managed policies

AWS managed policies

An *AWS managed policy* is a standalone policy that is created and administered by AWS. A *standalone policy* means that the policy has its own Amazon Resource Name (ARN) that includes the policy name. For example, `arn:aws:iam::aws:policy/IAMReadOnlyAccess` is an AWS managed policy. For more information about ARNs, see IAM ARNs. For a list of AWS managed policies for AWS services, see AWS managed policies.

AWS managed policies make it convenient for you to assign appropriate permissions to users, IAM groups, and roles. It is faster than writing the policies yourself, and includes permissions for many common use cases.

You cannot change the permissions defined in AWS managed policies. AWS occasionally updates the permissions defined in an AWS managed policy. When AWS does this, the update affects all principal entities (IAM users, IAM groups, and IAM roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API calls become available for existing services. For example, the AWS managed policy called **ReadOnlyAccess** provides read-only access to all AWS services and resources. When AWS launches a new service, AWS updates the **ReadOnlyAccess** policy to add read-only permissions for the new service. The updated permissions are applied to all principal entities that the policy is attached to.

Full access AWS managed policies: These define permissions for service administrators by granting full access to a service. Examples include:

- AmazonDynamoDBFullAccess
- IAMFullAccess

Power-user AWS managed policies: These provide full access to AWS services and resources, but do not allow managing users and IAM groups. Examples include:

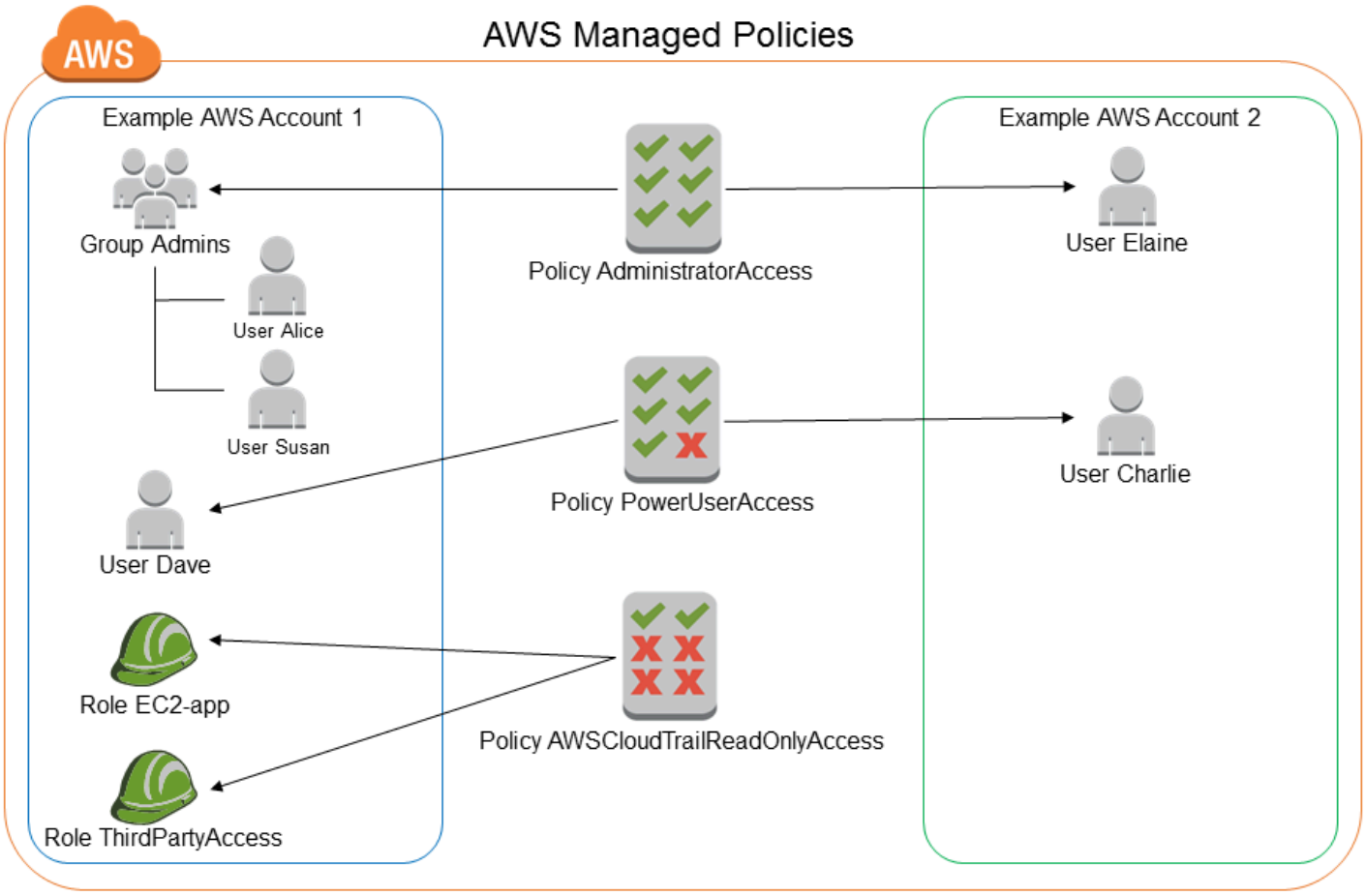
- AWSCodeCommitPowerUser
- AWSKeyManagementServicePowerUser

Partial-access AWS managed policies: These provide specific levels of access to AWS services without allowing permissions management access level permissions. Examples include:

- AmazonMobileAnalyticsWriteOnlyAccess
- AmazonEC2ReadOnlyAccess

Job function AWS managed policies: These policies align closely with commonly used job functions in the IT industry and facilitate granting permissions for these job functions. One key advantage of using job function policies is that they are maintained and updated by AWS as new services and API operations are introduced. For example, the AdministratorAccess job function provides full access and permissions delegation to every service and resource in AWS. We recommend that you use this policy only for the account administrator. For power users that require full access to every service except limited access to IAM and AWS Organizations, use the PowerUserAccess job function. For a list and descriptions of the job function policies, see AWS managed policies for job functions.

The following diagram illustrates AWS managed policies. The diagram shows three AWS managed policies: **AdministratorAccess**, **PowerUserAccess**, and **AWSCloudTrail_ReadOnlyAccess**. Notice that a single AWS managed policy can be attached to principal entities in different AWS accounts, and to different principal entities in a single AWS account.



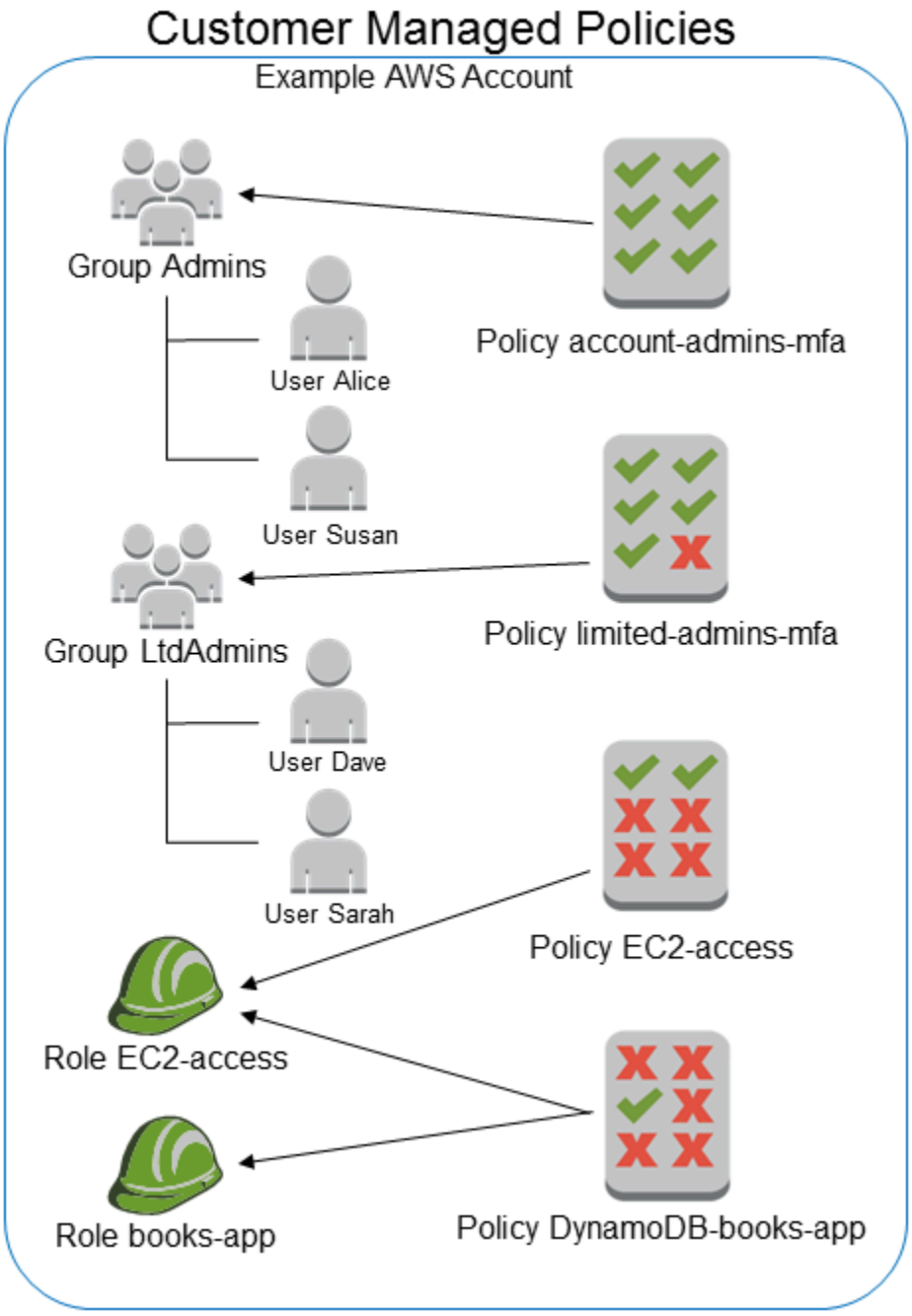
Customer managed policies

You can create standalone policies in your own AWS account that you can attach to principal entities (IAM users, IAM groups, and IAM roles). You create these *customer managed policies* for your specific use cases, and you can change and update them as often as you like. Like AWS managed policies, when you attach a policy to a principal entity, you give the entity the permissions that are defined in the policy. When you update permissions in the policy, the changes are applied to all principal entities that the policy is attached to.

A great way to create a customer managed policy is to start by copying an existing AWS managed policy. That way you know that the policy is correct at the beginning and all you need to do is customize it to your environment.

The following diagram illustrates customer managed policies. Each policy is an entity in IAM with its own Amazon Resource Name (ARN) that includes the policy name. Notice that the same policy can be attached to multiple principal entities—for example, the same **DynamoDB-books-app** policy is attached to two different IAM roles.

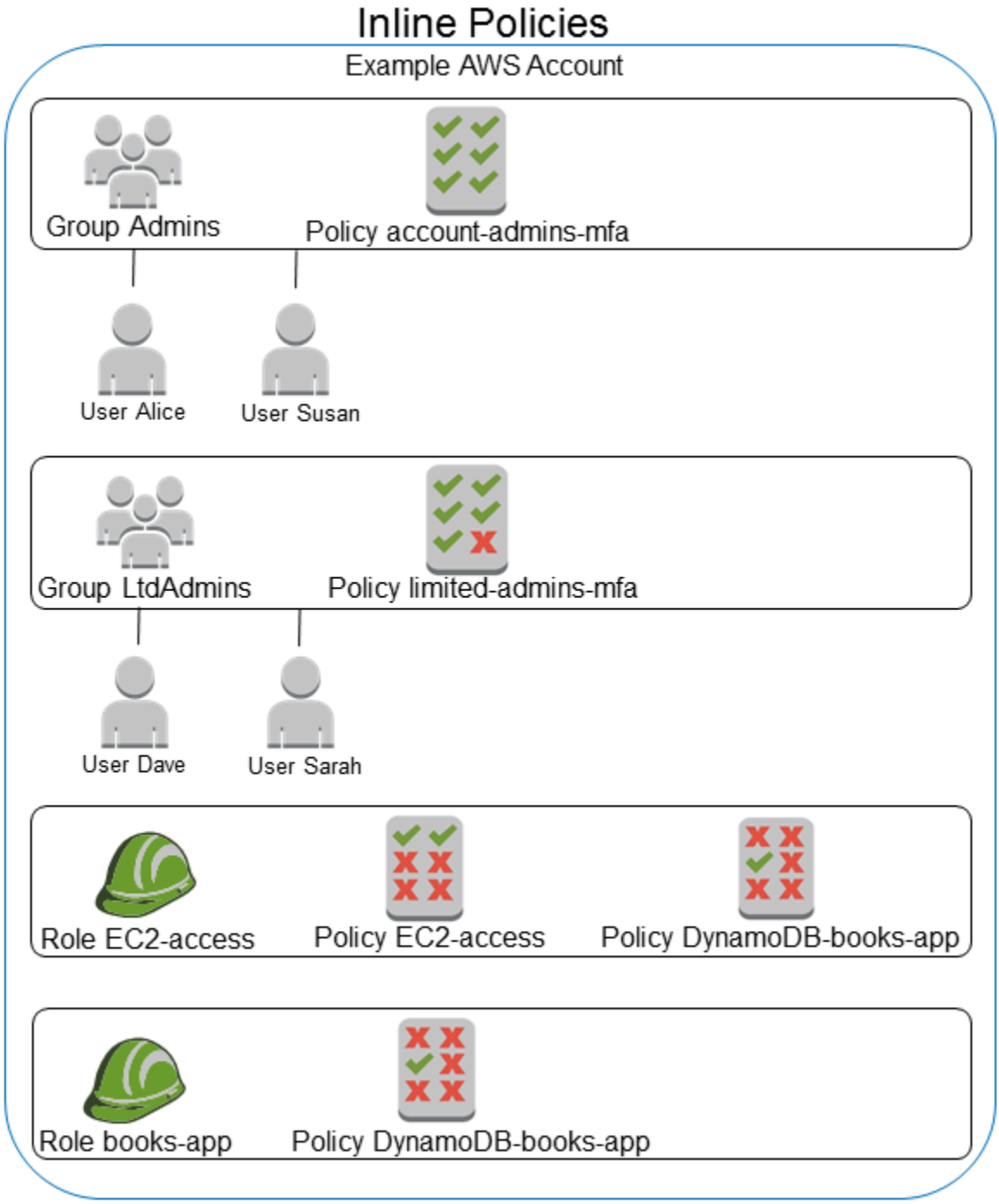
For more information, see Define custom IAM permissions with customer managed policies



Inline policies

An inline policy is a policy created for a single IAM identity (a user, user group, or role). Inline policies maintain a strict one-to-one relationship between a policy and an identity. They are deleted when you delete the identity. You can create a policy and embed it in an identity, either when you create the identity or later. If a policy could apply to more than one entity, it's better to use a managed policy.

The following diagram illustrates inline policies. Each policy is an inherent part of the user, group, or role. Notice that two roles include the same policy (the **DynamoDB-books-app** policy), but they are not sharing a single policy. Each role has its own copy of the policy.



Choose between managed policies and inline policies

Consider your use cases when deciding between managed and inline policies. In most cases, we recommend that you use managed policies instead of inline policies.

> **Note**
>
> You can use both managed and inline policies together to define common and unique permissions for a principal entity.

Managed policies provide the following features:

Reusability

A single managed policy can be attached to multiple principal entities (users, groups, and roles). You can create a library of policies that define useful permissions for your AWS account, and then attach these policies to principal entities as needed.

Central change management

When you change a managed policy, the change is applied to all principal entities that the policy is attached to. For example, if you want to add permission for a new AWS API, you can update a customer managed policy or associate an AWS managed policy to add the permission. If you're using an AWS managed policy, AWS updates the policy. When a managed policy is updated, the changes are applied to all principal entities that the managed policy is attached to. In contrast, to change an inline policy, you must individually edit each identity that contains the inline policy. For example, if a group and a role both contain the same inline policy, you must individually edit both principal entities to change that policy.

Versioning and rolling back

When you change a customer managed policy, the changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. IAM stores up to five versions of your customer managed policies. You can use policy versions to revert a policy to an earlier version as needed.

> **Note**
>
> A policy version is different from a `Version` policy element. The `Version` policy element is used within a policy and defines the version of the policy language. To learn

more about policy versions, see the section called “Versioning IAM policies”. To learn more about the `Version` policy element see IAM JSON policy elements: Version.

Delegating permissions management

You can allow users in your AWS account to attach and detach policies while maintaining control over the permissions defined in those policies. To do this, designate some users as full administrators—that is, administrators that can create, update, and delete policies. You can then designate other users as limited administrators. Those limited administrators can attach policies to other principal entities, but only the policies that you have allowed them to attach.

For more information about delegating permissions management, see Controlling access to policies.

Larger policy character limits

The maximum character size limit for managed policies is greater than the character limit for group inline policies. If you reach the inline policy's character size limit, you can create more IAM groups and attach the managed policy to the group.

For more information on quotas and limits, see IAM and AWS STS quotas.

Automatic updates for AWS managed policies

AWS maintains AWS managed policies and updates them when necessary, for example, to add permissions for new AWS services, without you having to make changes. The updates are automatically applied to the principal entities that you have attached the AWS managed policy to.

Get started with managed policies

We recommend using policies that grant least privilege, or granting only the permissions required to perform a task. The most secure way to grant least privilege is to write a customer managed policy with only the permissions needed by your team. You must create a process to allow your team to request more permissions when necessary. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need.

To get started adding permissions to your IAM identities (users, groups of users, and roles), you can use AWS managed policies. AWS managed policies don't grant least privilege permissions. You

must consider the security risk of granting your principals more permissions than they need to do their job.

You can attach AWS managed policies, including job functions, to any IAM identity. For more information, see Adding and removing IAM identity permissions.

To switch to least privilege permissions, you can run AWS Identity and Access Management and Access Analyzer to monitor the principals with AWS managed policies. After learning which permissions they are using, then you can write or generate a customer managed policy with only the required permissions for your team. This is less secure, but provides more flexibility as you learn how your team is using AWS. For more information, see IAM Access Analyzer policy generation.

AWS managed policies are designed to provide permissions for many common use cases. For more information about AWS managed policies that are designed for specific job functions, see AWS managed policies for job functions.

For a list of AWS managed policies, see AWS Managed Policy Reference Guide.

Using inline policies

Inline policies are useful if you want to maintain a strict one-to-one relationship between a policy and the identity to which it is applied. For example, if you want to be sure that the permissions in a policy are not inadvertently assigned to an identity other than the one they're intended for. When you use an inline policy, the permissions in the policy cannot be inadvertently attached to the wrong identity. In addition, when you use the AWS Management Console to delete that identity, the policies embedded in the identity are deleted as well because they are part of the principal entity.

Convert an inline policy to a managed policy

If you have inline policies in your account, you can convert them to managed policies. To do this, copy the policy to a new managed policy. Next, attach the new policy to the identity that has the inline policy. Then delete the inline policy.

Converting an inline policy to a managed policy

To convert an inline policy to a managed policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **User groups**, **Users**, or **Roles**.
3. In the list, choose the name of the user group, user, or role that has the policy you want to remove.
4. Choose the **Permissions** tab.
5. For IAM groups, select the name of the inline policy that you want to remove. For users and roles, choose **Show *n* more**, if necessary, and then expand the inline policy that you want to remove.
6. Choose **Copy** to copy the JSON policy document for the policy.
7. In the navigation pane, choose **Policies**.
8. Choose **Create policy** and then choose the **JSON** option.
9. Replace the existing text with your JSON policy text, and then choose **Next**.
10. Enter a name and optional description for your policy and choose **Create policy**.
11. In the navigation pane, choose **User groups**, **Users**, or **Roles**, and again choose the name of the user group, user, or role that has the policy you want to remove.
12. Choose the **Permissions** tab and then choose **Add permissions**.
13. For IAM groups, select the checkbox next to the name of your new policy, choose **Add permissions**, and then choose **Attach policy**. For users or roles, choose **Add permissions**. On the next page, choose **Attach existing policies directly**, select the checkbox next to the name of your new policy, choose **Next**, and then choose **Add permissions**.

    You are returned to the **Summary** page for your user group, user, or role.
14. Select the checkbox next to the inline policy that you want to remove and choose **Remove**.

Deprecated AWS managed policies

To simplify the assignment of permissions, AWS provides managed policies—predefined policies that are ready to be attached to your IAM users, groups, and roles.

Sometimes AWS needs to add a new permission to an existing policy, such as when a new service is introduced. Adding a new permission to an existing policy does not disrupt or remove any feature or ability.

However, AWS might choose to create a *new* policy when the needed changes could impact customers if they were applied to an existing policy. For example, removing permissions from an

existing policy could break the permissions of any IAM entity or application that depended upon it, potentially disrupting a critical operation.

Therefore, when such a change is required, AWS creates a completely new policy with the required changes and makes it available to customers. The old policy is then marked *deprecated*. For more information, see Deprecated AWS managed policies in *AWS Managed Policy Reference Guide*.

Establish permissions guardrails using data perimeters

Data perimeter guardrails are meant to serve as always-on boundaries to help protect your data across a broad set of AWS accounts and resources. Data perimeters follow IAM security best practices to establish permissions guardrails across multiple accounts. These organization-wide permissions guardrails do not replace your existing fine-grained access controls. Instead, they work as **coarse-grained access controls** that help improve your security strategy by ensuring users, roles, and resources adhere to a set of defined security standards.

A data perimeter is set of permission guardrails in your AWS environment which help make sure that only your trusted identities are accessing trusted resources from expected networks.

- Trusted identities: Principals (IAM roles or users) in your AWS accounts and AWS services acting on your behalf.
- Trusted resources: Resources owned by your AWS accounts or by AWS services acting on your behalf.
- Expected networks: Your on-premises data centers and virtual private clouds (VPCs), or networks of AWS services acting on your behalf.

> **Note**
>
> In some cases, you might need to extend your data perimeter to also include access by your trusted business partners. You should consider all intended data access patterns when you create a definition of trusted identities, trusted resources, and expected networks specific to your company and your use of AWS services.

Data perimeter controls should be treated as any other security control within the information security and risk management program. This means that you should perform a threat analysis to identify potential risks within your cloud environment, and then, based on your own risk acceptance criteria, select and implement appropriate data perimeter controls. To better inform

the iterative risk-based approach to data perimeter implementation, you need to understand what security risks and threat vectors are addressed by data perimeter controls as well as your security priorities.

Data perimeter controls

Data perimeter coarse-grained controls help you achieve six distinct security objectives across three data perimeters through the implementation of different combinations of Policy types and condition keys.

Perimeter	Control objective	Using	Applied on	Global condition context keys
Identity	Only trusted identities can access my resources	RCP	Resources	aws:PrincipalOrgID aws:PrincipalOrgPaths
	Only trusted identities are allowed from my network	VPC endpoint policy	Network	aws:PrincipalAccount aws:PrincipalIsAwsService aws:SourceOrgID aws:SourceOrgPath aws:SourceAccount
Resources	Your identities can access only trusted resources	SCP	Identities	aws:ResourceOrgID

Perimeter	Control objective	Using	Applied on	Global condition context keys
	Only trusted resources can be accessed from your network	VPC endpoint policy	Network	aws:ResourceOrgPaths aws:ResourceAccount
Network	Your identities can access resources only from expected networks	SCP	Identities	aws:SourceIp aws:SourceVpc aws:SourceVpce aws:VpceAccount aws:VpceOrgPaths aws:VpceOrgID aws:ViaAWSService aws:PrincipalIsAwsService
	Your resources can only be accessed from expected networks	RCP	Resources	

You can think of data perimeters as creating a firm boundary around your data to prevent unintended access patterns. Although data perimeters can prevent broad unintended access, you still need to make fine-grained access control decisions. Establishing a data perimeter does not diminish the need to continuously fine-tune permissions by using tools such as IAM Access Analyzer as part of your journey to least privilege.

To enforce data perimeter controls on resources that are currently not supported by RCPs, you can use resource-based policies that are attached to resources directly. For a list of services that support RCPs and resource-based policies, see Resource control policies (RCPs) and AWS services that work with IAM.

To enforce network perimeter controls, we recommend that you use `aws:VpceOrgID`, `aws:VpceOrgPaths`, and `aws:VpceAccount` only if all services you want to restrict access to are currently supported. Using these condition keys with unsupported services can lead to unintended authorization results. For a list of services that support the keys, see the AWS global condition context keys. If you need to enforce the controls on a wider range of services, consider using `aws:SourceVpc` and `aws:SourceVpce` instead.

Identity perimeter

An identity perimeter is a set of coarse-grained preventative access controls that help make sure only trusted identities can access your resources and only trusted identities are allowed from your network. Trusted identities usually include principals (roles or users) in your AWS accounts and AWS services acting on your behalf. All other identities are considered untrusted and are prevented by the identity perimeter unless an explicit exception is granted.

The following global condition keys help enforce identity perimeter controls based on your definition of trusted identities. Use these keys in resource control policies to restrict access to resources, or in VPC endpoint policies to restrict access to your networks.

Identities owned by you

You can use the following condition keys to define IAM principals that you create and manage in your AWS accounts.

- aws:PrincipalOrgID – You can use this condition key to make sure that IAM principals making the request belong to the specified organization in AWS Organizations.
- aws:PrincipalOrgPaths – You can use this condition key to make sure that the IAM user , IAM role, AWS STS federated user principal, SAML federated principal, OIDC federated principal, or AWS account root user making the request belong to the specified organizational unit (OU) in AWS Organizations.
- aws:PrincipalAccount – You can use this condition key to make sure resources can only be accessed by the principal account that you specify in the policy.

Identities of AWS services acting on your behalf

You can use the following condition keys to allow AWS services to use their own identities to access your resources when they act on your behalf.

- aws:PrincipalIsAWSService and aws:SourceOrgID (or aws:SourceOrgPaths and aws:SourceAccount) – You can use these condition keys to make sure that when AWS service principals access your resources, they do it only on behalf of a resource in the specified organization, organizational unit, or an account in AWS Organizations.

For more information, see Establishing a data perimeter on AWS: Allow only trusted identities to access company data.

Resource perimeter

A resource perimeter is a set of coarse-grained preventative access controls that help make sure your identities can access only trusted resources and only trusted resources can be accessed from your network. Trusted resources usually include resources owned by your AWS accounts or by AWS services acting on your behalf.

The following global condition keys help enforce resource perimeter controls based on your definition of trusted resources. Use these keys in Service control policies (SCPs) to restrict which resources can be accessed by your identities, or in VPC endpoint policies to restrict which resources can be accessed from your networks.

Resources owned by you

You can use the following condition keys to define AWS resources that you create and manage in your AWS accounts.

- aws:ResourceOrgID – You can use this condition key to make sure the resource that is being accessed belongs to the specified organization in AWS Organizations.
- aws:ResourceOrgPaths – You can use this condition key to make sure the resource that is being accessed belongs to the specified organizational unit(OU) in AWS Organizations.
- aws:ResourceAccount – You can use this condition key to make sure the resource that is being accessed belongs to the specified AWS account.

Resources of AWS services acting on your behalf

In some cases, you might need to permit access to AWS owned resources, resources that do not belong to your organization and that are accessed by your principals or by AWS services acting on your behalf. For more information about these scenarios, see Establishing a data perimeter on AWS: Allow only trusted resources from my organization.

Network perimeter

A network perimeter is a set of coarse-grained preventative access controls that help make sure your identities can access resources only from expected networks and your resources can only be accessed from expected networks. Expected networks usually include your on-premises data centers and virtual private clouds (VPCs) and networks of AWS services acting on your behalf.

The following global condition keys help enforce network perimeter controls based on your definition of expected networks. Use these keys in service control policies (SCPs) to restrict networks your identities can communicate from, or in resource control policies (RCPs) to constrain resource access to expected networks.

Networks owned by you

You can use the following condition keys to define networks your employees and applications are expected to use to access your resources, such as your corporate IP CIDR range and your VPCs.

- aws:SourceIp – You can use this condition key to make sure the requester's IP address is within a specified IP range.
- aws:SourceVpc – You can use this condition key to make sure the VPC endpoint the request travels through belongs to the specified VPC.
- aws:SourceVpce – You can use this condition key to make sure the request travels through the specified VPC endpoint.
- aws:VpceAccount – You can use this condition key to make sure requests come through VPC endpoints owned by the specified AWS account.
- aws:VpceOrgPaths – You can use this condition key to make sure that requests come through VPC endpoints owned by accounts that belong to the specified organizational unit (OU) in AWS Organizations.
- aws:VpceOrgID – You can use this condition key to make sure requests come through VPC endpoints owned by accounts in the specified organization in AWS Organizations.

`aws:VpceAccount`, `aws:VpceOrgPaths`, and `aws:VpceOrgID` are particularly useful for implementing network perimeter controls that scale automatically with your VPC endpoint usage, without requiring updates to policies when you create new endpoints. See the AWS global condition context keys for the list of AWS services that support these keys.

Networks of AWS services acting on your behalf

You can use the following condition keys to allow AWS services to access your resources from their networks when they act on your behalf.

- aws:ViaAWSService – You can use this condition key to make sure that AWS services can make requests on behalf of your principal using Forward access sessions (FAS).
- aws:PrincipalIsAWSService – You can use this condition key to make sure that AWS services can access your resources using AWS service principals.

There are additional scenarios where you need to permit access to AWS services that access your resources from outside your network. For more information, see Establishing a data perimeter on AWS: Allow access to company data only from expected networks.

Resources to learn more about data perimeters

The following resources can help you learn more about data perimeters across AWS.

- Data perimeters on AWS – Learn about data perimeters and their benefits and use cases.
- Blog Post Series: Establishing a Data Perimeter on AWS – These blog posts cover prescriptive guidance about establishing your data perimeter at scale, including key security and implementation considerations.
- Data perimeter policy examples – This GitHub repository contains example policies that cover some common patterns to help you implement a data perimeter on AWS.
- Data perimeter helper – This tool helps you design and anticipate the impact of your data perimeter controls by analyzing access activity in your AWS CloudTrail logs.

Permissions boundaries for IAM entities

AWS supports *permissions boundaries* for IAM entities (users or roles). A permissions boundary is an advanced feature for using a managed policy to set the maximum permissions that an identity-based policy can grant to an IAM entity. An entity's permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

For more information about policy types, see Policy types.

Important

Don't use resource-based policy statements that include a `NotPrincipal` policy element with a `Deny` effect for IAM users or roles that have a permissions boundary policy attached. The `NotPrincipal` element with a `Deny` effect will always deny any IAM principal that has a permissions boundary policy attached, regardless of the values specified in the `NotPrincipal` element. This causes some IAM users or roles that would otherwise have access to the resource to lose access.
We recommend changing your resource-based policy statements to use the condition operator `ArnNotEquals` with the `aws:PrincipalArn` context key to limit access instead of the `NotPrincipal` element. For information about the `NotPrincipal` element, see AWS JSON policy elements: NotPrincipal.

You can use an AWS managed policy or a customer managed policy to set the boundary for an IAM entity (user or role). That policy limits the maximum permissions for the user or role.

For example, assume that the IAM user named `Shirley` should be allowed to manage only Amazon S3, Amazon CloudWatch, and Amazon EC2. To enforce this rule, you can use the following policy to set the permissions boundary for the `Shirley` user:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

When you use a policy to set the permissions boundary for a user, it limits the user's permissions but does not provide permissions on its own. In this example, the policy sets the maximum permissions of `Shirley` as all operations in Amazon S3, CloudWatch, and Amazon EC2. Shirley can never perform operations in any other service, including IAM, even if she has a permissions policy that allows it. For example, you can add the following policy to the `Shirley` user:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:CreateUser",
    "Resource": "*"
  }
}
```

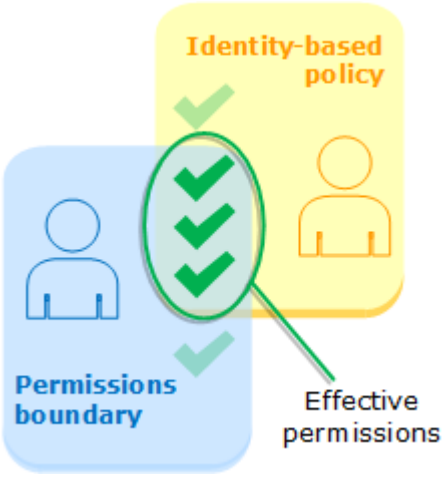
This policy allows creating a user in IAM. If you attach this permissions policy to the `Shirley` user, and Shirley tries to create a user, the operation fails. It fails because the permissions boundary does not allow the `iam:CreateUser` operation. Given these two policies, Shirley does not have permission to perform any operations in AWS. You must add a different permissions policy to allow actions in other services, such as Amazon S3. Alternatively, you could update the permissions boundary to allow her to create a user in IAM.

Evaluating effective permissions with boundaries

The permissions boundary for an IAM entity (user or role) sets the maximum permissions that the entity can have. This can change the effective permissions for that user or role. The effective permissions for an entity are the permissions that are granted by all the policies that affect the user or role. Within an account, the permissions for an entity can be affected by identity-based policies, resource-based policies, permissions boundaries, AWS Organizations SCPs, or session policies. For more information about the different types of policies, see Policies and permissions in AWS Identity and Access Management.

If any one of these policy types explicitly denies access for an operation, then the request is denied. The permissions granted to an entity by multiple permissions types are more complex. For more details about how AWS evaluates policies, see Policy evaluation logic.

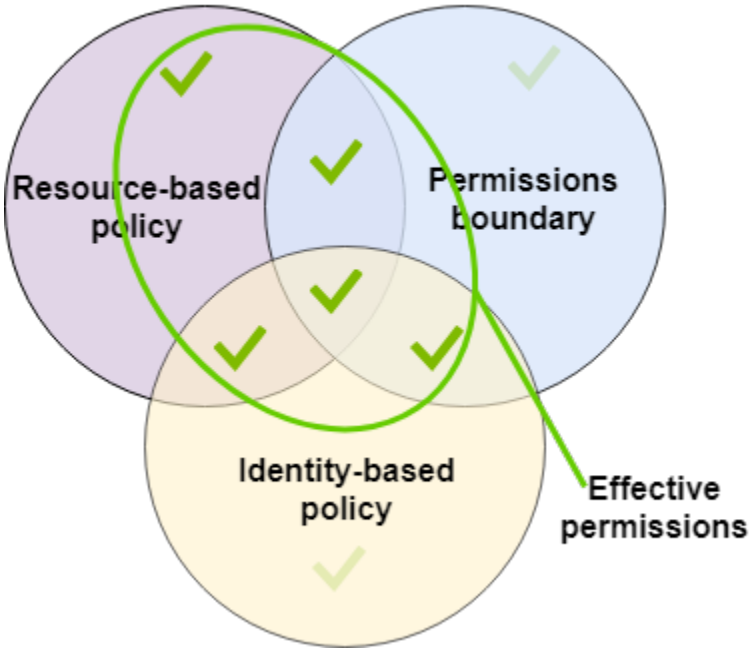
Identity-based policies with boundaries – Identity-based policies are inline or managed policies that are attached to a user, group of users, or role. Identity-based policies grant permission to the entity, and permissions boundaries limit those permissions. The effective permissions are the intersection of both policy types. An explicit deny in either of these policies overrides the allow.



Resource-based policies – Resource-based policies control how the specified principal can access the resource to which the policy is attached.

Resource-based policies for IAM users

Within the same account, resource-based policies that grant permissions to an IAM user ARN (that is not an AWS STS federated user principal session) are not limited by an implicit deny in an identity-based policy or permissions boundary.



Resource-based policies for IAM roles

IAM role – Resource-based policies that grant permissions to an IAM role ARN are limited by an implicit deny in a permissions boundary or session policy.

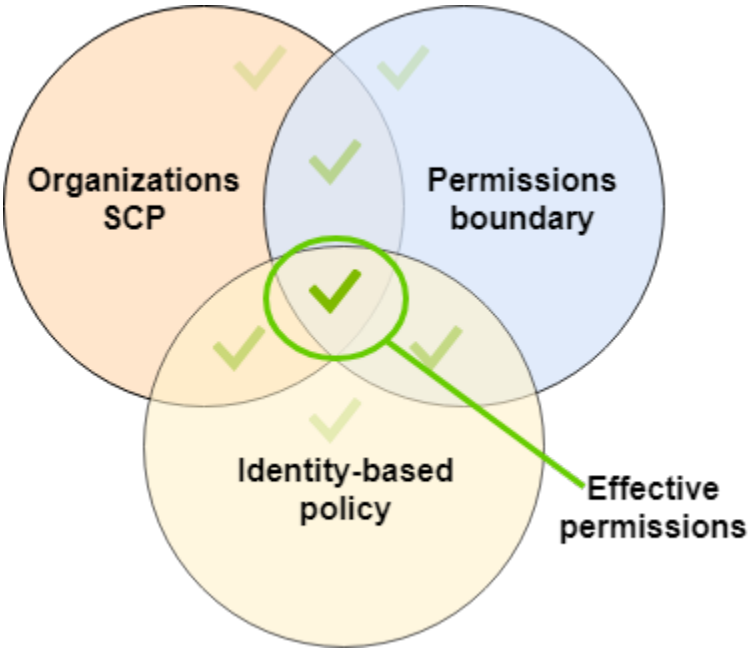
IAM role session – Within the same account, resource-based policies that grant permissions to an IAM role session ARN grant permissions directly to the assumed role session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy. When you assume a role and make a request, the principal making the request is the IAM role session ARN and not the ARN of the role itself.

Resource-based policies for AWS STS federated user principal sessions

AWS STS federated user principal sessions – An AWS STS federated user principal session is a session created by calling `GetFederationToken`. When a federated user makes a request, the principal making the request is the federated user ARN and not the ARN of the IAM user who federated. Within the same account, resource-based policies that grant permissions to a federated user ARN grant permissions directly to the session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy.

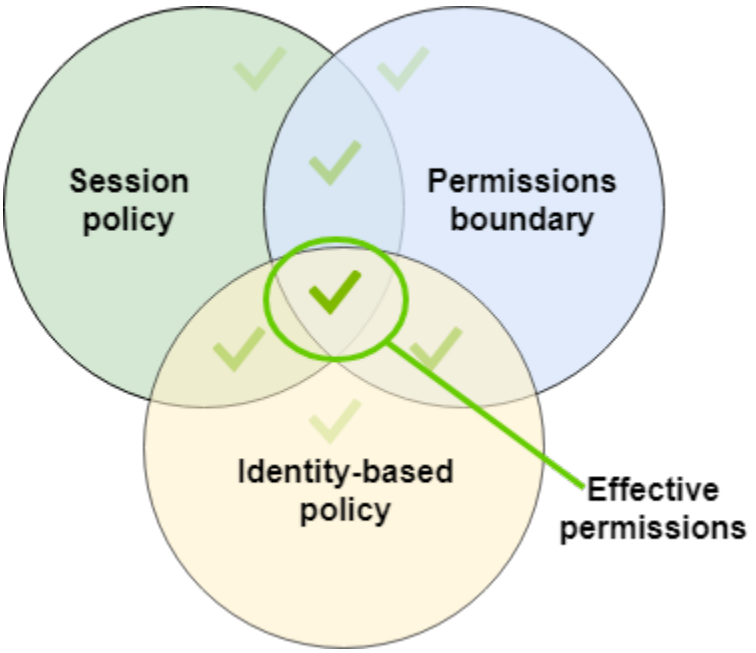
However, if a resource-based policy grants permission to the ARN of the IAM user who federated, then requests made by the AWS STS federated user principal during the session are limited by an implicit deny in a permission boundary or session policy.

AWS Organizations SCPs – SCPs are applied to an entire AWS account. They limit permissions for every request made by a principal within the account. An IAM entity (user or role) can make a request that is affected by an SCP, a permissions boundary, and an identity-based policy. In this case, the request is allowed only if all three policy types allow it. The effective permissions are the intersection of all three policy types. An explicit deny in any of these policies overrides the allow.



You can learn whether your account is a member of an organization in AWS Organizations. Organization members might be affected by an SCP. To view this data using the AWS CLI command or AWS API operation, you must have permissions for the `organizations:DescribeOrganization` action for your AWS Organizations entity. You must have additional permissions to perform the operation in the AWS Organizations console. To learn whether an SCP is denying access to a specific request, or to change your effective permissions, contact your AWS Organizations administrator.

Session policies – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The permissions for a session come from the IAM entity (user or role) used to create the session and from the session policy. The entity's identity-based policy permissions are limited by the session policy and the permissions boundary. The effective permissions for this set of policy types are the intersection of all three policy types. An explicit deny in any of these policies overrides the allow. For more information about session policies, see Session Policies.



Delegating responsibility to others using permissions boundaries

You can use permissions boundaries to delegate permissions management tasks, such as user creation, to IAM users in your account. This permits others to perform tasks on your behalf within a specific boundary of permissions.

For example, assume that María is the administrator of the X-Company AWS account. She wants to delegate user creation duties to Zhang. However, she must ensure that Zhang creates users that adhere to the following company rules:

- Users cannot use IAM to create or manage users, groups, roles, or policies.
- Users are denied access to the Amazon S3 `logs` bucket and cannot access the `i-1234567890abcdef0` Amazon EC2 instance.
- Users cannot remove their own boundary policies.

To enforce these rules, María completes the following tasks, for which details are included below:

1. María creates the `XCompanyBoundaries` managed policy to use as a permissions boundary for all new users in the account.
2. María creates the `DelegatedUserBoundary` managed policy and assigns it as the permissions boundary for Zhang. Maria makes a note of her admin user's ARN and uses it in the policy to prevent Zhang from accessing it.

3. María creates the `DelegatedUserPermissions` managed policy and attaches it as a permissions policy for Zhang.
4. María tells Zhang about his new responsibilities and limitations.

Task 1: María must first create a managed policy to define the boundary for the new users. María will allow Zhang to give users the permissions policies they need, but she wants those users to be restricted. To do this, she creates the following customer managed policy with the name `XCompanyBoundaries`. This policy does the following:

- Allows users full access to several services
- Allows limited self-managing access in the IAM console. This means they can change their password after signing into the console. They can't set their initial password. To allow this, add the `"*LoginProfile"` action to the `AllowManageOwnPasswordAndAccessKeys` statement.
- Denies users access to the Amazon S3 logs bucket or the `i-1234567890abcdef0` Amazon EC2 instance

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ServiceBoundaries",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:*",
                "dynamodb:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMConsoleForCredentials",
            "Effect": "Allow",
            "Action": [
                "iam:ListUsers",
                "iam:GetAccountPasswordPolicy"
```

```
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswordAndAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:*AccessKey*",
                "iam:ChangePassword",
                "iam:GetUser",
                "iam:*ServiceSpecificCredential*",
                "iam:*SigningCertificate*"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::logs",
                "arn:aws:s3:::logs/*"
            ]
        },
        {
            "Sid": "DenyEC2Production",
            "Effect": "Deny",
            "Action": "ec2:*",
            "Resource": "arn:aws:ec2:*:*:instance/i-1234567890abcdef0"
        }
    ]
}
```

Each statement serves a different purpose:

1. The `ServiceBoundaries` statement of this policy allows full access to the specified AWS services. This means that a new user's actions in these services are limited only by the permissions policies that are attached to the user.
2. The `AllowIAMConsoleForCredentials` statement allows access to list all IAM users. This access is necessary to navigate the **Users** page in the AWS Management Console. It also allows

viewing the password requirements for the account, which is necessary when changing your own password.

3. The `AllowManageOwnPasswordAndAccessKeys` statement allows users to manage only their own console password and programmatic access keys. This is important if Zhang or another administrator assigns a new user a permissions policy with full IAM access. In that case, that user could then change their own or other users' permissions. This statement prevents that from happening.
4. The `DenyS3Logs` statement explicitly denies access to the `logs` bucket.
5. The `DenyEC2Production` statement explicitly denies access to the `i-1234567890abcdef0` instance.

Task 2: María wants to allow Zhang to create all X-Company users, but only with the `XCompanyBoundaries` permissions boundary. She creates the following customer managed policy named `DelegatedUserBoundary`. This policy defines the maximum permissions that Zhang can have.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateOrChangeOnlyWithBoundary",
            "Effect": "Allow",
            "Action": [
                "iam:AttachUserPolicy",
                "iam:CreateUser",
                "iam:DeleteUserPolicy",
                "iam:DetachUserPolicy",
                "iam:PutUserPermissionsBoundary",
                "iam:PutUserPolicy"
            ],
            "Resource": "*",
            "Condition": {
               "StringEquals": {
                 "iam:PermissionsBoundary": "arn:aws:iam::123456789012:policy/
XCompanyBoundaries"
                }
            }
```

```
        },
        {
            "Sid": "CloudWatchAndOtherIAMTasks",
            "Effect": "Allow",
            "Action": [
              "cloudwatch:*",
              "iam:CreateAccessKey",
              "iam:CreateGroup",
              "iam:CreateLoginProfile",
              "iam:CreatePolicy",
              "iam:DeleteGroup",
              "iam:DeletePolicy",
              "iam:DeletePolicyVersion",
              "iam:DeleteUser",
              "iam:GetAccountPasswordPolicy",
              "iam:GetGroup",
              "iam:GetLoginProfile",
              "iam:GetPolicy",
              "iam:GetPolicyVersion",
              "iam:GetRolePolicy",
              "iam:GetUser",
              "iam:GetUserPolicy",
              "iam:ListAccessKeys",
              "iam:ListAttachedRolePolicies",
              "iam:ListAttachedUserPolicies",
              "iam:ListEntitiesForPolicy",
              "iam:ListGroups",
              "iam:ListGroupsForUser",
              "iam:ListMFADevices",
              "iam:ListPolicies",
              "iam:ListPolicyVersions",
              "iam:ListRolePolicies",
              "iam:ListSSHPublicKeys",
              "iam:ListServiceSpecificCredentials",
              "iam:ListSigningCertificates",
              "iam:ListUserPolicies",
              "iam:ListUsers",
              "iam:SetDefaultPolicyVersion",
              "iam:SimulateCustomPolicy",
              "iam:SimulatePrincipalPolicy",
              "iam:UpdateGroup",
              "iam:UpdateLoginProfile",
              "iam:UpdateUser"
            ],
```

```
            "NotResource": "arn:aws:iam::123456789012:user/Maria"
        },
        {
            "Sid": "NoBoundaryPolicyEdit",
            "Effect": "Deny",
            "Action": [
                "iam:CreatePolicyVersion",
                "iam:DeletePolicy",
                "iam:DeletePolicyVersion",
                "iam:SetDefaultPolicyVersion"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:policy/XCompanyBoundaries",
                "arn:aws:iam::123456789012:policy/DelegatedUserBoundary"
            ]
        },
        {
            "Sid": "NoBoundaryUserDelete",
            "Effect": "Deny",
            "Action": "iam:DeleteUserPermissionsBoundary",
            "Resource": "*"
        }
    ]
}
```

Each statement serves a different purpose:

1. The `CreateOrChangeOnlyWithBoundary` statement allows Zhang to create IAM users but only if he uses the `XCompanyBoundaries` policy to set the permissions boundary. This statement also allows him to set the permissions boundary for existing users but only using that same policy. Finally, this statement allows Zhang to manage permissions policies for users with this permissions boundary set.
2. The `CloudWatchAndOtherIAMTasks` statement allows Zhang to complete other user, group, and policy management tasks. He has permissions to reset passwords and create access keys for any IAM user not listed in the `NotResource` policy element. This allows him to help users with sign-in issues.
3. The `NoBoundaryPolicyEdit` statement denies Zhang access to update the `XCompanyBoundaries` policy. He is not allowed to change any policy that is used to set the permissions boundary for himself or other users.

4. The `NoBoundaryUserDelete` statement denies Zhang access to delete the permissions boundary for himself or other users.

María then assigns the `DelegatedUserBoundary` policy as the permissions boundary for the Zhang user.

Task 3: Because the permissions boundary limits the maximum permissions, but does not grant access on its own, Maria must create a permissions policy for Zhang. She creates the following policy named `DelegatedUserPermissions`. This policy defines the operations that Zhang can perform, within the defined boundary.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "IAM",
            "Effect": "Allow",
            "Action": "iam:*",
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchLimited",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetDashboard",
                "cloudwatch:GetMetricData",
                "cloudwatch:ListDashboards",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3BucketContents",
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::ZhangBucket"
        }
    ]
```

```
}
```

Each statement serves a different purpose:

1. The `IAM` statement of the policy allows Zhang full access to IAM. However, because his permissions boundary allows only some IAM operations, his effective IAM permissions are limited only by his permissions boundary.
2. The `CloudWatchLimited` statement allows Zhang to perform five actions in CloudWatch. His permissions boundary allows all actions in CloudWatch, so his effective CloudWatch permissions are limited only by his permissions policy.
3. The `S3BucketContents` statement allows Zhang to list the `ZhangBucket` Amazon S3 bucket. However, his permissions boundary does not allow any Amazon S3 action, so he cannot perform any S3 operations, regardless of his permissions policy.

> **Note**
>
> Zhang's policies allow him to create a user that can then access Amazon S3 resources that he can't access. By delegating these administrative actions, Maria effectively trusts Zhang with access to Amazon S3.

María then attaches the `DelegatedUserPermissions` policy as the permissions policy for the `Zhang` user.

Task 4: She gives Zhang instructions to create a new user. She tells him that he can create new users with any permissions that they need, but he must assign them the `XCompanyBoundaries` policy as a permissions boundary.

Zhang completes the following tasks:

1. Zhang creates a user with the AWS Management Console. He types the user name `Nikhil` and enables console access for the user. He clears the checkbox next to **Requires password reset**, because the policies above allow users to change their passwords only after they are signed in to the IAM console.
2. On the **Set permissions** page, Zhang chooses the **IAMFullAccess** and **AmazonS3ReadOnlyAccess** permissions policies that allow Nikhil to do his work.
3. Zhang skips the **Set permissions boundary** section, forgetting María's instructions.

4. Zhang reviews the user details and chooses **Create user**.

   The operation fails and access is denied. Zhang's `DelegatedUserBoundary` permissions boundary requires that any user he creates have the `XCompanyBoundaries` policy used as a permissions boundary.

5. Zhang returns to the previous page. In the **Set permissions boundary** section, he chooses the `XCompanyBoundaries` policy.

6. Zhang reviews the user details and chooses **Create user**.

   The user is created.

When Nikhil signs in, he has access to IAM and Amazon S3, except those operations that are denied by the permissions boundary. For example, he can change his own password in IAM but can't create another user or edit his policies. Nikhil has read-only access to Amazon S3.

If someone adds a resource-based policy to the `logs` bucket that allows Nikhil to put an object in the bucket, he still cannot access the bucket. The reason is that any actions on the `logs` bucket are explicitly denied by his permissions boundary. An explicit deny in any policy type results in a request being denied.

However, if a resource-based policy attached to a Secrets Manager secret allows Nikhil to perform the `secretsmanager:GetSecretValue` action, then Nikhil can retrieve and decrypt the secret. The reason is that Secrets Manager operations are not explicitly denied by his permissions boundary, and implicit denies in permissions boundaries do not limit resource-based policies.

Identity-based policies and resource-based policies

A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. When you create a permissions policy to restrict access to a resource, you can choose an *identity-based policy* or a *resource-based policy*.

Identity-based policies are attached to an IAM user, group, or role. These policies let you specify what that identity can do (its permissions). For example, you can attach the policy to the IAM user named John, stating that he is allowed to perform the Amazon EC2 `RunInstances` action. The policy could further state that John is allowed to get items from an Amazon DynamoDB table named `MyCompany`. You can also allow John to manage his own IAM security credentials. Identity-based policies can be managed or inline.

Resource-based policies are attached to a resource. For example, you can attach resource-based policies to Amazon S3 buckets, Amazon SQS queues, VPC endpoints, AWS Key Management Service encryption keys, Amazon DynamoDB tables and streams, and AWS Sign-In resources. For a list of services that support resource-based policies, see AWS services that work with IAM.

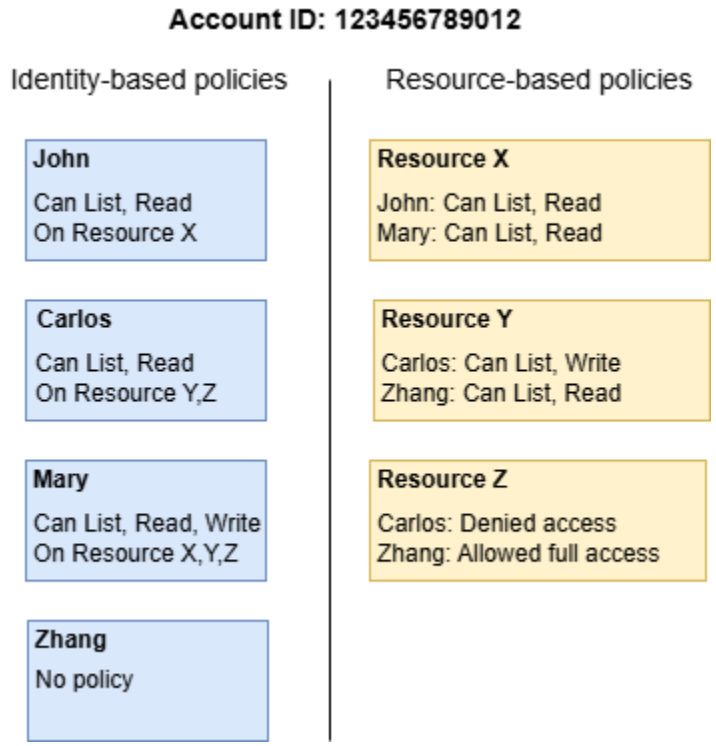
With resource-based policies, you can specify who has access to the resource and what actions they can perform on it. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?. Resource-based policies are inline only, not managed.

> **Note**
>
> *Resource-based* policies differ from *resource-level* permissions. You can attach resource-based policies directly to a resource, as described in this topic. Resource-level permissions refer to the ability to use ARNs to specify individual resources in a policy. Resource-based policies are supported only by some AWS services. For a list of which services support resource-based policies and resource-level permissions, see AWS services that work with IAM.

To learn how identity-based policies and resource-based policies interact within the same account, see Policy evaluation for requests within a single account.

To learn how the policies interact across accounts, see Cross-account policy evaluation logic.

To better understand these concepts, view the following figure. The administrator of the `123456789012` account attached *identity-based policies* to the `John`, `Carlos`, and `Mary` users. Some of the actions in these policies can be performed on specific resources. For example, the user `John` can perform some actions on `Resource X`. This is a *resource-level permission* in an identity-based policy. The administrator also added *resource-based policies* to `Resource X`, `Resource Y`, and `Resource Z`. Resource-based policies allow you to specify who can access that resource. For example, the resource-based policy on `Resource X` allows the `John` and `Mary` users list and read access to the resource.



The 123456789012 account example allows the following users to perform the listed actions:

- **John** – John can perform list and read actions on `Resource X`. He is granted this permission by the identity-based policy on his user and the resource-based policy on `Resource X`.
- **Carlos** – Carlos can perform list, read, and write actions on `Resource Y`, but is denied access to `Resource Z`. The identity-based policy on Carlos allows him to perform list and read actions on `Resource Y`. The `Resource Y` resource-based policy also allows him write permissions. However, although his identity-based policy allows him access to `Resource Z`, the `Resource Z` resource-based policy denies that access. An explicit `Deny` overrides an `Allow` and his access to `Resource Z` is denied. For more information, see Policy evaluation logic.
- **Mary** – Mary can perform list, read, and write operations on `Resource X`, `Resource Y`, and `Resource Z`. Her identity-based policy allows her more actions on more resources than the resource-based policies, but none of them deny access.
- **Zhang** – Zhang has full access to `Resource Z`. Zhang has no identity-based policies, but the `Resource Z` resource-based policy allows him full access to the resource. Zhang can also perform list and read actions on `Resource Y`.

Identity-based policies and resource-based policies are both permissions policies and are evaluated together. For a request to which only permissions policies apply, AWS first checks all policies for

a Deny. If one exists, then the request is denied. Then AWS checks for each Allow. If at least one policy statement allows the action in the request, the request is allowed. It doesn't matter whether the Allow is in the identity-based policy or the resource-based policy.

> **Important**
>
> This logic applies only when the request is made within a single AWS account. For requests made from one account to another, the requester in Account A must have an identity-based policy that allows them to make a request to the resource in Account B. Also, the resource-based policy in Account B must allow the requester in Account A to access the resource. There must be policies in both accounts that allow the operation, otherwise the request fails. For more information about using resource-based policies for cross-account access, see Cross account resource access in IAM.

A user who has specific permissions might request a resource that also has a permissions policy attached to it. In that case, AWS evaluates both sets of permissions when determining whether to grant access to the resource. For information about how policies are evaluated, see Policy evaluation logic.

> **Note**
>
> Amazon S3 supports identity-based policies and resource-based policies (referred to as *bucket policies*). In addition, Amazon S3 supports a permission mechanism known as an *access control list (ACL)* that is independent of IAM policies and permissions. You can use IAM policies in combination with Amazon S3 ACLs. For more information, see Access Control in the *Amazon Simple Storage Service User Guide*.

Control access to AWS resources using policies

You can use a policy to control access to resources within IAM or all of AWS.

To use a policy to control access in AWS, you must understand how AWS grants access. AWS is composed of collections of *resources*. An IAM user is a resource. An Amazon S3 bucket is a resource. When you use the AWS API, the AWS CLI, or the AWS Management Console to perform an operation (such as creating a user), you send a *request* for that operation. Your request specifies

an action, a resource, a *principal entity* (user or role), a *principal account*, and any necessary request information. All of this information provides *context*.

AWS then checks that you (the principal) are authenticated (signed in) and authorized (have permission) to perform the specified action on the specified resource. During authorization, AWS checks all the policies that apply to the context of your request. Most policies are stored in AWS as JSON documents and specify the permissions for principal entities. For more information about policy types and uses, see Policies and permissions in AWS Identity and Access Management.

AWS authorizes the request only if each part of your request is allowed by the policies. To view a diagram of this process, see How IAM works. For details about how AWS determines whether a request is allowed, see Policy evaluation logic.

When you create an IAM policy, you can control access to the following:

- **Principals** – Control what the person making the request (the principal) is allowed to do.
- **IAM Identities** – Control which IAM identities (IAM groups, users, and roles) can be accessed and how.
- **IAM Policies** – Control who can create, edit, and delete customer managed policies, and who can attach and detach all managed policies.
- **AWS Resources** – Control who has access to resources using an identity-based policy or a resource-based policy.
- **AWS Accounts** – Control whether a request is allowed only for members of a specific account.

Policies let you specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM user starts with no permissions. In other words, by default, users can do nothing, not even view their own access keys. To give a user permission to do something, you can add the permission to the user (that is, attach a policy to the user). Or you can add the user to a user group that has the intended permission.

For example, you might grant a user permission to list his or her own access keys. You might also expand that permission and also let each user create, update, and delete their own keys.

When you give permissions to a user group, all users in that user group get those permissions. For example, you can give the Administrators user group permission to perform any of the IAM actions on any of the AWS account resources. Another example: You can give the Managers user group permission to describe the Amazon EC2 instances of the AWS account.

For information about how to delegate basic permissions to your users, IAM groups, and roles, see Permissions required to access IAM resources. For additional examples of policies that illustrate basic permissions, see Example policies for administering IAM resources.

Controlling access for principals

You can use policies to control what the person making the request (the principal) is allowed to do. To do this, you must attach an identity-based policy to that person's identity (user, user group, or role). You can also use a permissions boundary to set the maximum permissions that an entity (user or role) can have.

For example, assume that you want the user Zhang Wei to have full access to CloudWatch, Amazon DynamoDB, Amazon EC2, and Amazon S3. You can create two different policies so that you can later break them up if you need one set of permissions for a different user. Or you can put both the permissions together in a single policy, and then attach that policy to the IAM user that is named Zhang Wei. You could also attach a policy to a user group to which Zhang belongs, or a role that Zhang can assume. As a result, when Zhang views the contents of an S3 bucket, his requests are allowed. If he tries to create a new IAM user, his request is denied because he doesn't have permission.

You can use a permissions boundary on Zhang to make sure that he is never given access to the `amzn-s3-demo-bucket1` S3 bucket. To do this, determine the *maximum* permissions that you want Zhang to have. In this case, you control what he does using his permissions policies. Here, you only care that he doesn't access the confidential bucket. So you use the following policy to define Zhang's boundary to allow all AWS actions for Amazon S3 and a few other services but deny access to the `amzn-s3-demo-bucket1` S3 bucket. Because the permissions boundary does not allow any IAM actions, it prevents Zhang from deleting his (or anyone's) boundary.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionsBoundarySomeServices",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:*",
                "dynamodb:*",
```

```
                "ec2:*",
                "s3:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PermissionsBoundaryNoConfidentialBucket",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1",
                "arn:aws:s3:::amzn-s3-demo-bucket1/*"
            ]
        }
    ]
}
```

When you assign a policy like this as a permissions boundary for a user, remember that it does not grant any permissions. It sets the maximum permissions that an identity-based policy can grant to an IAM entity. For more information about permissions boundaries, see Permissions boundaries for IAM entities.

For detailed information about the procedures mentioned previously, refer to these resources:

- To learn more about creating an IAM policy that you can attach to a principal, see Define custom IAM permissions with customer managed policies.
- To learn how to attach an IAM policy to a principal, see Adding and removing IAM identity permissions.
- To see an example policy for granting full access to EC2, see Amazon EC2: Allows full EC2 access within a specific Region, programmatically and in the console.
- To allow read-only access to an S3 bucket, use the first two statements of the following example policy: Amazon S3: Allows read and write access to objects in an S3 Bucket, programmatically and in the console.
- To see an example policy for allowing users to set their credentials, such as their console password, their programmatic access keys, and their MFA devices, see AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page.

Controlling access to identities

You can use IAM policies to control what your users can do to an identity by creating a policy that you attach to all users through a user group. To do this, create a policy that limits what can be done to an identity, or who can access it.

For example, you can create a user group named **AllUsers**, and then attach that user group to all users. When you create the user group, you might give all your users access to set their credentials as described in the previous section. You can then create a policy that denies access to change the user group unless the user name is included in the condition of the policy. But that part of the policy only denies access to anyone except those users listed. You also have to include permissions to allow all the user group management actions for everyone in the user group. Finally, you attach this policy to the user group so that it is applied to all users. As a result, when a user not specified in the policy tries to make changes to the user group, the request is denied.

To create this policy with the visual editor

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. Choose **Create policy**.
4. On the **Policy editor** section, choose the **Visual** option.
5. In **Select a service** choose **IAM**.
6. In **Actions allowed**, type **`group`** in the search box. The visual editor shows all the IAM actions that contain the word `group`. Select all of the checkboxes.
7. Choose **Resources** to specify resources for your policy. Based on the actions you chose, you should see **group** and **user** resource types.

   - **group** – Choose **Add ARNs**. For **Resource in**, select the **Any account** option. Select the **Any group name with path** checkbox and then type the user group name **`AllUsers`**. Then choose **Add ARNs**.
   - **user** – Select the checkbox next to **Any in this account**.

One of the actions that you chose, `ListGroups`, does not support using specific resources. You do not have to choose **All resources** for that action. When you save your policy or view the policy in the **JSON** editor, you can see that IAM automatically creates a new permission block granting this action permission on all resources.

8. To add another permission block, choose **Add more permissions**.
9. Choose **Select a service** and then choose **IAM**.
10. Choose **Actions allowed** and then choose **Switch to deny permissions**. When you do that, the entire block is used to deny permissions.
11. Type **`group`** in the search box. The visual editor shows you all the IAM actions that contain the word `group`. Select the check boxes next to the following actions:

    - **CreateGroup**
    - **DeleteGroup**
    - **RemoveUserFromGroup**
    - **AttachGroupPolicy**
    - **DeleteGroupPolicy**
    - **DetachGroupPolicy**
    - **PutGroupPolicy**
    - **UpdateGroup**
12. Choose **Resources** to specify the resources for your policy. Based on the actions that you chose, you should see the **group** resource type. Choose **Add ARNs**. For **Resource in**, select the **Any account** option. For **Any group name with path**, type the user group name **`AllUsers`**. Then choose **Add ARNs**.
13. Choose **Request conditions - *optional*** and then choose **Add another condition**. Complete the form with the following values:

    - **Condition key** – Choose **aws:username**
    - **Qualifier** – Choose **Default**
    - **Operator** – Choose **StringNotEquals**
    - **Value** – Type **`srodriguez`** and then choose **Add** to add another value. Type **`mjackson`** and then choose **Add** to add another value. Type **`adesai`** and then choose **Add condition**.

This condition ensures that access will be denied to the specified user group management actions when the user making the call is not included in the list. Because this explicitly denies permission, it overrides the previous block that allowed those users to call the actions. Users on the list are not denied access, and they are granted permission in the first permission block, so they can fully manage the user group.

14. When you are finished, choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor option, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

15. On the **Review and create** page, for the **Policy Name**, type **`LimitAllUserGroupManagement`**. For the **Description**, type **`Allows all users read-only access to a specific user group, and allows only specific users access to make changes to the user group`**. Review **Permissions defined in this policy** to make sure that you have granted the intended permissions. Then choose **Create policy** to save your new policy.
16. Attach the policy to your user group. For more information, see Adding and removing IAM identity permissions.

Alternatively, you can create the same policy using this example JSON policy document. To view this JSON policy, see IAM: Allows specific IAM users to manage a group programmatically and in the console. For detailed instructions for creating a policy using a JSON document, see the section called “Creating policies using the JSON editor”.

Controlling access to policies

You can control how your users can apply AWS managed policies. To do this, attach this policy to all your users. Ideally, you can do this using a user group.

For example, you might create a policy that allows users to attach only the IAMUserChangePassword and PowerUserAccess AWS managed policies to a new IAM user, user group, or role.

For customer managed policies, you can control who can create, update, and delete these policies. You can control who can attach and detach policies to and from principal entities (IAM groups, users, and roles). You can also control which policies a user can attach or detach, and to and from which entities.

For example, you can give permissions to an account administrator to create, update, and delete policies. Then you give permissions to a team leader or other limited administrator to attach and detach these policies to and from principal entities that the limited administrator manages.

For more information, refer to these resources:

- To learn more about creating an IAM policy that you can attach to a principal, see Define custom IAM permissions with customer managed policies.
- To learn how to attach an IAM policy to a principal, see Adding and removing IAM identity permissions.
- To see an example policy for limiting the use of managed policies, see IAM: Limits managed policies that can be applied to an IAM user, group, or role.

Controlling permissions for creating, updating, and deleting customer managed policies

You can use IAM policies to control who is allowed to create, update, and delete customer managed policies in your AWS account. The following list contains API operations that pertain directly to creating, updating, and deleting policies or policy versions:

- CreatePolicy
- CreatePolicyVersion
- DeletePolicy
- DeletePolicyVersion
- SetDefaultPolicyVersion

The API operations in the preceding list correspond to actions that you can allow or deny—that is, permissions that you can grant—using an IAM policy.

Consider the following example policy. It allows a user to create, update (that is, create a new policy version), delete, and set a default version for all customer managed policies in the AWS account. The example policy also allows the user to list policies and get policies. To learn how to

create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

Example policy that allows creating, updating, deleting, listing, getting, and setting the default version for all policies

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "iam:CreatePolicy",
      "iam:CreatePolicyVersion",
      "iam:DeletePolicy",
      "iam:DeletePolicyVersion",
      "iam:GetPolicy",
      "iam:GetPolicyVersion",
      "iam:ListPolicies",
      "iam:ListPolicyVersions",
      "iam:SetDefaultPolicyVersion"
    ],
    "Resource": "*"
  }
}
```

You can create policies that limit the use of these API operations to affect only the managed policies that you specify. For example, you might want to allow a user to set the default version and delete policy versions, but only for specific customer managed policies. You do this by specifying the policy ARN in the `Resource` element of the policy that grants these permissions.

The following example shows a policy that allows a user to delete policy versions and set the default version. But these actions are only allowed for the customer managed policies that include the path /TEAM-A/. The customer managed policy ARN is specified in the `Resource` element of the policy. (In this example the ARN includes a path and a wildcard and thus matches all customer managed policies that include the path /TEAM-A/). To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

For more information about using paths in the names of customer managed policies, see Friendly names and paths.

Example policy that allows deleting policy versions and setting the default version for only specific policies

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:DeletePolicyVersion",
            "iam:SetDefaultPolicyVersion"
        ],
        "Resource": "arn:aws:iam::111122223333:policy/TEAM-A/*"
    }
}
```

Controlling permissions for attaching and detaching managed policies

You can also use IAM policies to allow users to work with only specific managed policies. In effect, you can control which permissions a user is allowed to grant to other principal entities.

The following list shows API operations that pertain directly to attaching and detaching managed policies to and from principal entities:

- AttachGroupPolicy
- AttachRolePolicy
- AttachUserPolicy
- DetachGroupPolicy
- DetachRolePolicy
- DetachUserPolicy

You can create policies that limit the use of these API operations to affect only the specific managed policies and/or principal entities that you specify. For example, you might want to allow

a user to attach managed policies, but only the managed policies that you specify. Or, you might want to allow a user to attach managed policies, but only to the principal entities that you specify.

The following example policy allows a user to attach managed policies to only the IAM groups and roles that include the path /TEAM-A/. The user group and role ARNs are specified in the `Resource` element of the policy. (In this example the ARNs include a path and a wildcard character and thus match all IAM groups and roles that include the path /TEAM-A/). To learn how to create a policy using this example JSON policy document, see the section called “Creating policies using the JSON editor”.

Example policy that allows attaching managed policies to only specific user groups or roles

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:AttachGroupPolicy",
            "iam:AttachRolePolicy"
        ],
        "Resource": [
            "arn:aws:iam::111122223333:group/TEAM-A/*",
            "arn:aws:iam::111122223333:role/TEAM-A/*"
        ]
    }
}
```

You can further limit the actions in the preceding example to affect only specific policies. That is, you can control which permissions a user is allowed to attach to other principal entities—by adding a condition to the policy.

In the following example, the condition ensures that the `AttachGroupPolicy` and `AttachRolePolicy` permissions are allowed only when the policy being attached matches one of the specified policies. The condition uses the `iam:PolicyARN` condition key to determine which policy or policies are allowed to be attached. The following example policy expands on the previous example. It allows a user to attach only the managed policies that include the path /TEAM-A/ to only the IAM groups and roles that include the path /TEAM-A/. To learn how to create

a policy using this example JSON policy document, see the section called "Creating policies using the JSON editor".

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:AttachGroupPolicy",
            "iam:AttachRolePolicy"
        ],
        "Resource": [
            "arn:aws:iam::111122223333:group/TEAM-A/*",
            "arn:aws:iam::111122223333:role/TEAM-A/*"
        ],
        "Condition": {
            "ArnLike": {
                "iam:PolicyARN": "arn:aws:iam::111122223333:policy/TEAM-A/*"
            }
        }
    }
}
```

This policy uses the `ArnLike` condition operator, but you can also use the `ArnEquals` condition operator because these two condition operators behave identically. For more information about `ArnLike` and `ArnEquals`, see Amazon Resource Name (ARN) condition operators in the *Condition Types* section of the *Policy Element Reference*.

For example, you can limit the use of actions to involve only the managed policies that you specify. You do this by specifying the policy ARN in the `Condition` element of the policy that grants these permissions. For example, to specify the ARN of a customer managed policy:

```
"Condition": {"ArnEquals":
  {"iam:PolicyARN": "arn:aws:iam::123456789012:policy/POLICY-NAME"}
}
```

You can also specify the ARN of an AWS managed policy in a policy's `Condition` element. The ARN of an AWS managed policy uses the special alias `aws` in the policy ARN instead of an account ID, as in this example:

```
"Condition": {"ArnEquals":
  {"iam:PolicyARN": "arn:aws:iam::aws:policy/AmazonEC2FullAccess"}
}
```

Controlling access to resources

You can control access to resources using an identity-based policy or a resource-based policy. In an identity-based policy, you attach the policy to an identity and specify what resources that identity can access. In a resource-based policy, you attach a policy to the resource that you want to control. In the policy, you specify which principals can access that resource. For more information about both types of policies, see Identity-based policies and resource-based policies.

For more information, refer to these resources:

- To learn more about creating an IAM policy that you can attach to a principal, see Define custom IAM permissions with customer managed policies.
- To learn how to attach an IAM policy to a principal, see Adding and removing IAM identity permissions.
- Amazon S3 supports using resource-based policies on their buckets. For more information, see Bucket Policy Examples.

Resource Creators Do Not Automatically Have Permissions

If you sign in using the AWS account root user credentials, you have permission to perform any action on resources that belong to the account. However, this isn't true for IAM users. An IAM user might be granted access to create a resource, but the user's permissions, even for that resource, are limited to what's been explicitly granted. This means that just because you create a resource, such as an IAM role, you do not automatically have permission to edit or delete that role. Additionally, your permission can be revoked at any time by the account owner or by another user who has been granted access to manage your permissions.

Controlling access to principals in a specific account

You can directly grant IAM users in your own account access to your resources. If users from another account need access to your resources, you can create an IAM role. A role is an entity that

includes permissions but isn't associated with a specific user. Users from other accounts can then assume the role and access resources according to the permissions you've assigned to the role. For more information, see Access for an IAM user in another AWS account that you own.

> **Note**
>
> Some services support resource-based policies as described in Identity-based policies and resource-based policies (such as Amazon S3, Amazon SNS, and Amazon SQS). For those services, an alternative to using roles is to attach a policy to the resource (bucket, topic, or queue) that you want to share. The resource-based policy can specify the AWS account that has permissions to access the resource.

Controlling access to and for IAM users and roles using tags

Use the information in the following section to control who can access your IAM users and roles and what resources your users and roles can access. For more general information and examples of controlling access to other AWS resources, including other IAM resources, see Tags for AWS Identity and Access Management resources.

> **Note**
>
> For details about case sensitivity for tag keys and tag key values, see Case sensitivity.

Tags can be attached to the IAM *resource*, passed in the *request*, or attached to the *principal* that is making the request. An IAM user or role can be both a resource and principal. For example, you can write a policy that allows a user to list the groups for a user. This operation is allowed only if the user making the request (the principal) has the same `project=blue` tag as the user they're trying to view. In this example, the user can view the group membership for any user, including themselves, as long as they are working on the same project.

To control access based on tags, you provide tag information in the condition element of a policy. When you create an IAM policy, you can use IAM tags and the associated tag condition key to control access to any of the following:

- **Resource** – Control access to user or role resources based on their tags. To do this, use the **aws:ResourceTag/*key-name*** condition key to specify which tag key-value pair must be attached to the resource. For more information, see Controlling access to AWS resources.

- **Request** – Control what tags can be passed in an IAM request. To do this, use the **aws:RequestTag/*key-name*** condition key to specify what tags can be added, changed, or removed from an IAM user or role. This key is used the same way for IAM resources and other AWS resources. For more information, see Controlling access during AWS requests.
- **Principal** – Control what the person making the request (the principal) is allowed to do based on the tags that are attached to that person's IAM user or role. To do this, use the **aws:PrincipalTag/*key-name*** condition key to specify what tags must be attached to the IAM user or role before the request is allowed.
- **Any part of the authorization process** – Use the **aws:TagKeys** condition key to control whether specific tag keys can be used in a request or by a principal. In this case, the key value does not matter. This key behaves similarly for IAM and other AWS services. However, when you tag a user in IAM, this also controls whether the principal can make the request to any service. For more information, see Controlling access based on tag keys.

You can create an IAM policy using the visual editor, using JSON, or by importing an existing managed policy. For details, see Define custom IAM permissions with customer managed policies.

> **Note**
>
> You can also pass session tags when you assume an IAM role or federate a user. These are valid only for the length of the session.

Controlling access for IAM principals

You can control what the principal is allowed to do based on the tags attached to that person's identity.

This example shows how you might create an identity-based policy that allows any user in this account to view the group membership for any user, including themselves, as long as they are working on the same project. This operation is allowed only when the user's resource tag and the principal's tag have the same value for the tag key `project`. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:ListGroupsForUser",
            "Resource": "arn:aws:iam::111222333444:user/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/project":
 "${aws:PrincipalTag/project}"}
            }
        }]
}
```

Controlling access based on tag keys

You can use tags in your IAM policies to control whether specific tag keys can be used in a request or by a principal.

This example shows how you might create an identity-based policy that allows removing only the tag with the `temporary` key from users. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "iam:UntagUser",
        "Resource": "*",
        "Condition": {"ForAllValues:StringEquals": {"aws:TagKeys":
 ["temporary"]}}
    }]
}
```

Controlling access to AWS resources using tags

You can use tags to control access to your AWS resources that support tagging, including IAM resources. You can tag IAM users and roles to control what they can access. To learn how to tag IAM users and roles, see Tags for AWS Identity and Access Management resources. Additionally, you can control access to the following IAM resources: customer managed policies, IAM identity providers, instance profiles, server certificates, and virtual MFA devices. To view a tutorial for creating and testing a policy that allows IAM roles with principal tags to access resources with matching tags, see IAM tutorial: Define permissions to access AWS resources based on tags. Use the information in the following section to control access to other AWS resources, including IAM resources, without tagging IAM users or roles.

Before you use tags to control access to your AWS resources, you must understand how AWS grants access. AWS is composed of collections of *resources*. An Amazon EC2 instance is a resource. An Amazon S3 bucket is a resource. You can use the AWS API, the AWS CLI, or the AWS Management Console to perform an operation, such as creating a bucket in Amazon S3. When you do, you send a *request* for that operation. Your request specifies an action, a resource, a *principal entity* (user or role), a *principal account*, and any necessary request information. All of this information provides *context*.

AWS then checks that you (the principal entity) are authenticated (signed in) and authorized (have permission) to perform the specified action on the specified resource. During authorization, AWS checks all the policies that apply to the context of your request. Most policies are stored in AWS as JSON documents and specify the permissions for principal entities. For more information about policy types and uses, see Policies and permissions in AWS Identity and Access Management.

AWS authorizes the request only if each part of your request is allowed by the policies. To view a diagram and learn more about the IAM infrastructure, see How IAM works. For details about how IAM determines whether a request is allowed, see Policy evaluation logic.

Tags are another consideration in this process because tags can be attached to the *resource* or passed in the *request* to services that support tagging. To control access based on tags, you provide tag information in the condition element of a policy. To learn whether an AWS service supports controlling access using tags, see AWS services that work with IAM and look for the services that have **Yes** in the **ABAC** column. Choose the name of the service to view the authorization and access control documentation for that service.

You can then create an IAM policy that allows or denies access to a resource based on that resource's tag. In that policy, you can use tag condition keys to control access to any of the following:

- **Resource** – Control access to AWS service resources based on the tags on those resources. To do this, use the **aws:ResourceTag/*key-name*** condition key to determine whether to allow access to the resource based on the tags that are attached to the resource.
- **Request** – Control what tags can be passed in a request. To do this, use the **aws:RequestTag/*key-name*** condition key to specify what tag key-value pairs can be passed in a request to tag an AWS resource.
- **Any part of the authorization process** – Use the **aws:TagKeys** condition key to control whether specific tag keys can be in a request.

You can create an IAM policy visually, using JSON, or by importing an existing managed policy. For details, see Define custom IAM permissions with customer managed policies.

> **Note**
>
> Some services allow users to specify tags when they create the resource if they have permissions to use the action that creates the resource.

Controlling access to AWS resources

You can use conditions in your IAM policies to control access to AWS resources based on the tags on that resource. You can do this using the global `aws:ResourceTag/tag-key` condition key, or a service-specific key. Some services support only the service-specific version of this key and not the global version.

> **Warning**
>
> Do not try to control who can pass a role by tagging the role and then using the `ResourceTag` condition key in a policy with the `iam:PassRole` action. This approach does not have reliable results. For more information about permissions required to pass a role to a service, see Grant a user permissions to pass a role to an AWS service.

This example shows how you might create an identity-based policy that allows starting or stopping Amazon EC2 instances. These operations are allowed only if the instance tag `Owner` has the value of the user name. This policy defines permissions for programmatic and console access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:StartInstances",
                "ec2:StopInstances"
            ],
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:DescribeInstances",
            "Resource": "*"
        }
    ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard` attempts to start an Amazon EC2 instance, the instance must be tagged `Owner=richard` or `owner=richard`. Otherwise he will be denied access. The tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON policy elements: Condition.

This example shows how you might create an identity-based policy that uses the `team` principal tag in the resource ARN. The policy grants permission to delete Amazon Simple Queue Service queues, but only if the queue name starts with the team name followed by `-queue`. For example, `qa-queue` if `qa` is the team name for the `team` principal tag.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": {
        "Sid": "AllQueueActions",
        "Effect": "Allow",
        "Action": "sqs:DeleteQueue",
        "Resource": "arn:aws:sqs:us-east-2:111122223333:${aws:PrincipalTag/team}-
queue"
      }
}
```

Controlling access during AWS requests

You can use conditions in your IAM policies to control what tag key-value pairs can be passed in a request that applies tags to an AWS resource.

This example shows how you might create an identity-based policy that allows using the Amazon EC2 `CreateTags` action to attach tags to an instance. You can attach tags only if the tag contains the `environment` key and the `preprod` or `production` values. If you want, you can use the `ForAllValues` modifier with the `aws:TagKeys` condition key to indicate that only the key `environment` is allowed in the request. This stops users from including other keys, such as accidentally using `Environment` instead of `environment`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "ec2:CreateTags",
        "Resource": "arn:aws:ec2:*:*:instance/*",
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/environment": [
                    "preprod",
                    "production"
                ]
```

```
            },
            "ForAllValues:StringEquals": {"aws:TagKeys": "environment"}
        }
    }
}
```

Controlling access based on tag keys

You can use a condition in your IAM policies to control whether specific tag keys can be used in a request.

We recommend that when you use policies to control access using tags, you use the `aws:TagKeys` condition key. AWS services that support tags might allow you to create multiple tag key names that differ only by case, such as tagging an Amazon EC2 instance with `stack=production` and `Stack=test`. Key names are not case sensitive in policy conditions. This means that if you specify `"aws:ResourceTag/TagKey1": "Value1"` in the condition element of your policy, then the condition matches a resource tag key named either `TagKey1` or `tagkey1`, but not both. To prevent duplicate tags with a key that varies only by case, use the `aws:TagKeys` condition to define the tag keys that your users can apply, or use tag policies, available with AWS Organizations. For more information see Tag Policies in the *AWS Organizations User Guide*.

This example shows how you might create an identity-based policy that allows creating and tagging a Secrets Manager secret, but only with the tag keys `environment` or `cost-center`. The `Null` condition ensures that the condition evaluates to `false` if there are no tags in the request.

```
{
        "Effect": "Allow",
        "Action": [
            "secretsmanager:CreateSecret",
            "secretsmanager:TagResource"
        ],
        "Resource": "*",
        "Condition": {
            "Null": {
                "aws:TagKeys": "false"
            },
            "ForAllValues:StringEquals": {
                "aws:TagKeys": [
                    "environment",
                    "cost-center"
```

```
                ]
            }
        }
}
```

Cross account resource access in IAM

For some AWS services, you can grant cross-account access to your resources using IAM. To do this, you can attach a resource policy directly to the resource that you want to share, or use a role as a proxy.

To share the resource directly, the resource that you want to share must support resource-based policies. Unlike an identity-based policy for a role, a resource-based policy specifies who (which principal) can access that resource.

Use a role as a proxy when you want to access resources in another account that do not support resource-based policies.

For details about the differences between these policy types, see Identity-based policies and resource-based policies.

> **Note**
>
> IAM roles and resource-based policies delegate access across accounts only within a single partition. For example, you have an account in US West (N. California) in the standard `aws` partition. You also have an account in China in the `aws-cn` partition. You can't use a resource-based policy in your account in China to allow access for users in your standard AWS account.

Cross-account access using roles

Not all AWS services support resource-based policies. For these services, you can use cross-account IAM roles to centralize permission management when providing cross-account access to multiple services. A cross-account IAM role is an IAM role that includes a trust policy that allows IAM principals in another AWS account to assume the role. Put simply, you can create a role in one AWS account that delegates specific permissions to another AWS account.

For information about attaching a policy to an IAM identity, see Manage IAM policies.

> **Note**
>
> When a principal switches to a role to temporarily use the permissions of the role, they give up their original permissions and take on the permissions assigned to the role they've assumed.

Let's take a look at the overall process as it applies to APN Partner software that needs to access a customer account.

1. The customer creates an IAM role in their own account with a policy that allows access the Amazon S3 resources that the APN partner requires. In this example, the role name is `APNPartner`.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "s3:*",
               "Resource": [
                   "arn:aws:s3:::bucket-name"
               ]
           }
       ]
   }
   ```

2. Then, the customer specifies that the role can be assumed by the partner's AWS account by providing the APN Partner's AWS account ID in the trust policy for the `APNPartner` role.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
   ```

```
                "AWS": "arn:aws:iam::111122223333:role/APN-user-name"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

3. The customer gives the Amazon Resource Name (ARN) of the role to the APN partner. The ARN is the fully qualified name of the role.

```
arn:aws:iam::Customer-Account-ID:role/APNPartner
```

> **Note**
>
> We recommend using an external ID in multi-tenant situations. For details, see Access to AWS accounts owned by third parties.

4. When the APN Partner's software needs to access the customer's account, the software calls the AssumeRole API in the AWS Security Token Service with the ARN of the role in the customer's account. STS returns a temporary AWS credential that allows the software to do its work.

For another example of granting cross-account access using roles, see Access for an IAM user in another AWS account that you own. You can also follow the IAM tutorial: Delegate access across AWS accounts using IAM roles.

Cross-account access using resource-based policies

When an account accesses a resource through another account using a resource-based policy, the principal still works in the trusted account and does not have to give up their permissions to receive the role permissions. In other words, the principal continues to have access to resources in the trusted account while having access to the resource in the trusting account. This is useful for tasks such as copying information to or from the shared resource in the other account.

The principals that you can specify in a resource based policy include accounts, IAM users, AWS STS federated user principals, SAML federated principals, OIDC federated principals, IAM roles, assumed-role sessions, or AWS services. For more information, see Specifying a principal.

To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see Identifying resources shared with an external entity.

The following list includes some of the AWS services that support resource-based policies. For a complete list of the growing number of AWS services that support attaching permission policies to resources instead of principals, see AWS services that work with IAM and look for the services that have **Yes** in the **Resource Based** column.

- **Amazon S3 buckets** — The policy is attached to the bucket, but the policy controls access to both the bucket and the objects in it. For more information, see Bucket policies for Amazon S3 in the *Amazon Simple Storage Service User Guide*. In some cases, it may be best to use roles for cross-account access to Amazon S3. For more information, see the example walkthroughs in the *Amazon Simple Storage Service User Guide*.
- **Amazon Simple Notification Service (Amazon SNS) topics** — For more information, see Example cases for Amazon SNS access control in the *Amazon Simple Notification Service Developer Guide*.
- **Amazon Simple Queue Service (Amazon SQS) queues** – For more information, see Appendix: The Access Policy Language in the *Amazon Simple Queue Service Developer Guide*.

Resource-based policies to delegate AWS permissions

If a resource grants permissions to principals in your account, you can then delegate those permissions to specific IAM identities. Identities are users, groups of users, or roles in your account. You delegate permissions by attaching a policy to the identity. You can grant up to the maximum permissions that are allowed by the resource-owning account.

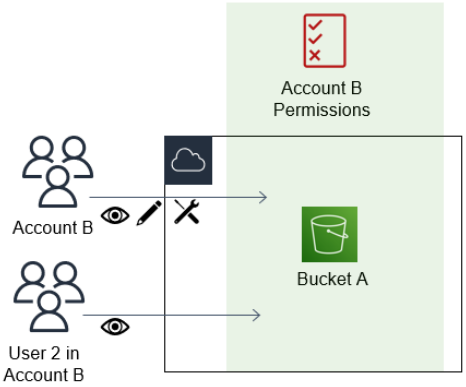
> **Important**
>
> In cross account access, a principal needs an `Allow` in the identity policy **and** the resource-based policy.

Assume that a resource-based policy allows all principals in your account full administrative access to a resource. Then you can delegate full access, read-only access, or any other partial access to principals in your AWS account. Alternatively, if the resource-based policy allows only list permissions, then you can delegate only list access. If you try to delegate more permissions than your account has, your principals will still have only list access.

For more information about how these decisions are made, see Determining whether a request is allowed or denied within an account.

> **Note**
>
> IAM roles and resource-based policies delegate access across accounts only within a single partition. For example, you can't add cross-account access between an account in the standard `aws` partition and an account in the `aws-cn` partition.

For example, assume that you manage `AccountA` and `AccountB`. In AccountA, you have an Amazon S3 bucket named `BucketA`.



1. You attach a resource-based policy to `BucketA` that allows all principals in AccountB full access to objects in your bucket. They can create, read, or delete any objects in that bucket.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "PrincipalAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::BucketA/*"
        }
```

```
    ]
}
```

AccountA gives AccountB full access to BucketA by naming AccountB as a principal in the resource-based policy. As a result, AccountB is authorized to perform any action on BucketA, and the AccountB administrator can delegate access to its users in AccountB.

The AccountB root user has all of the permissions that are granted to the account. Therefore, the root user has full access to BucketA.

2. In AccountB, attach a policy to the IAM user named User2. That policy allows the user read-only access to the objects in BucketA. That means that User2 can view the objects, but not create, edit, or delete them.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect" : "Allow",
            "Action" : [
                "s3:Get*",
                "s3:List*" ],
                "Resource" : "arn:aws:s3:::BucketA/*"
        }
    ]
}
```

   The maximum level of access that AccountB can delegate is the access level that is granted to the account. In this case, the resource-based policy granted full access to AccountB, but User2 is granted only read-only access.

   The AccountB administrator does not give access to User1. By default, users do not have any permissions except those that are explicitly granted, so User1 does not have access to BucketA.

IAM evaluates a principal's permissions at the time the principal makes a request. If you use wildcards (*) to give users full access to your resources, principals can access any resources that your

AWS account has access to. This is true even for resources you add or gain access to after creating the user's policy.

In the preceding example, if AccountB had attached a policy to User2 that allowed full access to all resources in all accounts, User2 would automatically have access to any resources that AccountB has access to. This includes the BucketA access and access to any other resources granted by resource-based policies in AccountA.

For more information about complex uses of roles, such as granting access to applications and services, see Common scenarios for IAM roles.

> **Important**
>
> Give access only to entities you trust, and give the minimum level of access necessary. Whenever the trusted entity is another AWS account, any IAM principal can be granted access to your resource. The trusted AWS account can delegate access only to the extent that it has been granted access; it cannot delegate more access than the account itself has been granted.

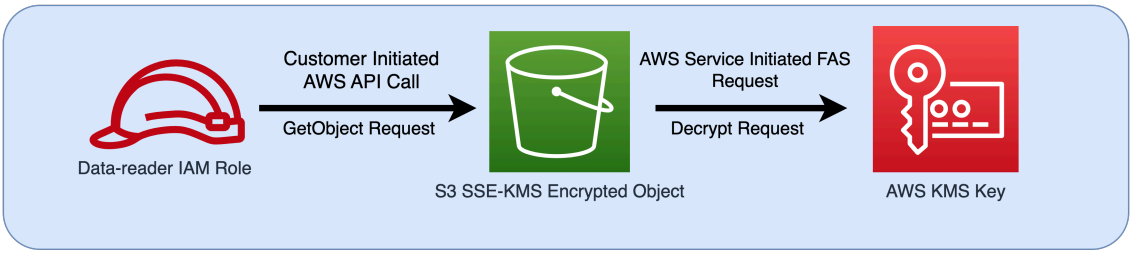
For information about permissions, policies, and the permission policy language that you use to write policies, see Access management for AWS resources.

Forward access sessions

Forward access sessions (FAS) is an IAM technology used by AWS services to pass your identity, permissions, and session attributes when an AWS service makes a request on your behalf. FAS uses the permissions of the identity calling an AWS service, combined with an AWS service's identity to make requests to downstream services. FAS requests are only made to AWS services on behalf of an IAM principal after a service has received a request that requires interactions with other AWS services or resources to complete. When a FAS request is made:

- The service that receives the initial request from an IAM principal checks the permissions of the IAM principal.
- The service that receives a subsequent FAS request also checks the permissions of the same IAM principal.

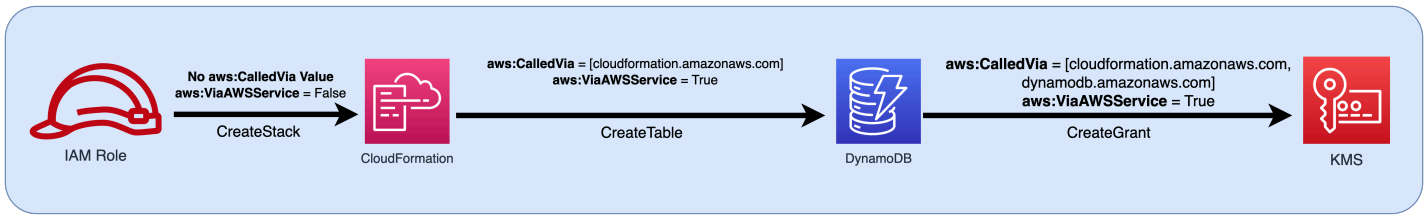
For example, FAS is used by Amazon S3 to make calls to AWS Key Management Service to decrypt an object when SSE-KMS was used to encrypt it. When downloading an SSE-KMS encrypted object, a role named **data-reader** calls GetObject on the object against Amazon S3, and does not call AWS KMS directly. After receiving the GetObject request and authorizing data-reader, Amazon S3 then makes a FAS request to AWS KMS in order to decrypt the Amazon S3 object. When KMS receives the FAS request it checks the permissions of the role and only authorizes the decryption request if data-reader has the correct permissions on the KMS key. The requests to both Amazon S3 and AWS KMS are authorized using the role's permissions and is only successful if data-reader has permissions to both the Amazon S3 object and the AWS KMS key.



> **Note**
>
> Additional FAS requests can be made by services who have received a FAS request. In such cases, the requesting principal must have permissions for all services called by FAS.

FAS Requests and IAM policy conditions

When FAS requests are made, aws:CalledVia, aws:CalledViaFirst, and aws:CalledViaLast condition keys are populated with the service principal of the service that initiated the FAS call. The aws:ViaAWSService condition key value is set to `true` whenever a FAS request is made. In the following diagram, the request to CloudFormation directly does not have any `aws:CalledVia` or `aws:ViaAWSService` condition keys set. When CloudFormation and DynamoDB make downstream FAS requests on the behalf of the role, the values for these condition keys are populated.



To allow a FAS request to be made when it would otherwise be denied by a Deny policy statement with a condition key testing Source IP addresses or Source VPCs, you must use condition keys to provide an exception for FAS requests in your Deny policy. This can be done for all FAS requests by using the `aws:ViaAWSService` condition key. To allow only specific AWS services to make FAS requests, use `aws:CalledVia`.

> **Important**
>
> When a FAS request is made after an initial request is made through a VPC endpoint, the condition key values for `aws:SourceVpce`, `aws:SourceVpc`, and `aws:VpcSourceIp` from the initial request are not used in FAS requests. When writing policies using `aws:VpcSourceIp` or `aws:SourceVpce` to conditionally grant access, you must also use `aws:ViaAWSService` or `aws:CalledVia` to allow FAS requests. When a FAS request is made after an initial request is received by a public AWS service endpoint, subsequent FAS requests will be made with the same `aws:SourceIP` condition key value.

Example: Allow Amazon S3 access from a VPC or with FAS

In the following IAM policy example, Amazon S3 GetObject and Athena requests are only allowed if they originate from VPC endpoints attached to *example_vpc*, or if the request is a FAS request made by Athena.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "OnlyAllowMyIPs",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject*",
        "athena:StartQueryExecution",
        "athena:GetQueryResults",
        "athena:GetWorkGroup",
        "athena:StopQueryExecution",
        "athena:GetQueryExecution"
      ],
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceVPC": [
          "vpc-111bbb22"
          ]
        }
      }
    },
    {
      "Sid": "OnlyAllowFAS",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "athena.amazonaws.com"
        }
      }
    }
  ]
}
```

For additional examples of using condition keys to allow FAS access, see the data perimeter example policy repo.

Example IAM identity-based policies

A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an IAM principal (user or role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents that are attached to an IAM identity (user, group of users, or role). Identity-based policies include AWS managed policies, customer managed policies, and inline policies. To learn how to create an IAM policy using these example JSON policy documents, see the section called “Creating policies using the JSON editor”.

By default all requests are denied, so you must provide access to the services, actions, and resources that you intend for the identity to access. If you also want to allow access to complete the specified actions in the IAM console, you need to provide additional permissions.

The following library of policies can help you define permissions for your IAM identities. After you find the policy that you need, choose **view this policy** to view the JSON for the policy. You can use the JSON policy document as a template for your own policies.

> **Note**
>
> If you would like to submit a policy to be included in this reference guide, use the **Feedback** button at the bottom of this page.

Example policies: AWS

- Allows access during a specific range of dates. (View this policy.)
- Allows enabling and disabling AWS Regions. (View this policy.)
- Allows MFA-authenticated users to manage their own credentials on the **Security credentials** page. (View this policy.)
- Allows specific access when using MFA during a specific range of dates. (View this policy.)
- Allows users to manage their own credentials on the **Security credentials** page. (View this policy.)
- Allows users to manage their own MFA device on the **Security credentials** page. (View this policy.)
- Allows users to manage their own password on the **Security credentials** page. (View this policy.)
- Allows users to manage their own password, access keys, and SSH public keys on the **Security credentials** page. (View this policy.)
- Denies access to AWS based on the requested Region. (View this policy.)
- Denies access to AWS based on the source IP address. (View this policy.)

Example policy: AWS Data Exchange

- Deny access to Amazon S3 resources outside of your account except AWS Data Exchange. (View this policy.)

Example policies: AWS Data Pipeline

- Denies access to pipelines that a user did not create (View this policy.)

Example policies: Amazon DynamoDB

- Allows access to a specific Amazon DynamoDB table (View this policy.)
- Allows access to specific Amazon DynamoDB attributes (View this policy.)
- Allows item-level access to Amazon DynamoDB based on an Amazon Cognito ID (View this policy.)

Example policies: Amazon EC2

- Allows attaching or detaching Amazon EBS volumes to Amazon EC2 instances based on tags (View this policy.)
- Allows launching Amazon EC2 instances in a specific subnet, programmatically and in the console (View this policy.)
- Allows managing Amazon EC2 security groups associated with a specific VPC, programmatically and in the console (View this policy.)
- Allows starting or stopping Amazon EC2 instances a user has tagged, programmatically and in the console (View this policy.)
- Allows starting or stopping Amazon EC2 instances based on resource and principal tags, programmatically and in the console (View this policy.)
- Allows starting or stopping Amazon EC2 instances when the resource and principal tags match (View this policy.)
- Allows full Amazon EC2 access within a specific Region, programmatically and in the console. (View this policy.)
- Allows starting or stopping a specific Amazon EC2 instance and modifying a specific security group, programmatically and in the console (View this policy.)
- Denies access to specific Amazon EC2 operations without MFA (View this policy.)
- Limits terminating Amazon EC2 instances to a specific IP address range (View this policy.)

Example policies: AWS Identity and Access Management (IAM)

- Allows access to the policy simulator API (View this policy.)
- Allows access to the policy simulator console (View this policy.)
- Allows assuming any roles that have a specific tag, programmatically and in the console (View this policy.)

- Allows and denies access to multiple services, programmatically and in the console (View this policy.)
- Allows adding a specific tag to an IAM user with a different specific tag, programmatically and in the console (View this policy.)
- Allows adding a specific tag to any IAM user or role, programmatically and in the console (View this policy.)
- Allows creating a new user only with specific tags (View this policy.)
- Allows generating and retrieving IAM credential reports (View this policy.)
- Allows managing a group's membership, programmatically and in the console (View this policy.)
- Allows managing a specific tag (View this policy.)
- Allows passing an IAM role to a specific service (View this policy.)
- Allows read-only access to the IAM console without reporting (View this policy.)
- Allows read-only access to the IAM console (View this policy.)
- Allows specific users to manage a group, programmatically and in the console (View this policy.)
- Allows setting the account password requirements, programmatically and in the console (View this policy.)
- Allows using the policy simulator API for users with a specific path (View this policy.)
- Allows using the policy simulator console for users with a specific path (View this policy.)
- Allows IAM users to self-manage an MFA device. (View this policy.)
- Allows IAM users to set their own credentials, programmatically and in the console. (View this policy.)
- Allows viewing service last accessed information for an AWS Organizations policy in the IAM console. (View this policy.)
- Limits managed policies that can be applied to an IAM user, group, or role (View this policy.)
- Allows access to IAM policies only in your account (View this policy.)

Example policies: AWS Lambda

- Allows an AWS Lambda function to access an Amazon DynamoDB table (View this policy.)

Example policies: Amazon RDS

- Allows full Amazon RDS database access within a specific Region. (View this policy.)

- Allows restoring Amazon RDS databases, programmatically and in the console (View this policy.)
- Allows tag owners full access to Amazon RDS resources that they have tagged (View this policy.)

Example policies: Amazon S3

- Allows an Amazon Cognito user to access objects in their own Amazon S3 bucket (View this policy.)
- Allows a user with temporary credentials to access their own home directory in Amazon S3, programmatically and in the console (View this policy.)
- Allows full S3 access, but explicitly denies access to the Production bucket if the administrator has not signed in using MFA within the last thirty minutes (View this policy.)
- Allows IAM users to access their own home directory in Amazon S3, programmatically and in the console (View this policy.)
- Allows a user to manage a single Amazon S3 bucket and denies every other AWS action and resource (View this policy.)
- Allows `Read` and `Write` access to a specific Amazon S3 bucket (View this policy.)
- Allows `Read` and `Write` access to a specific Amazon S3 bucket, programmatically and in the console (View this policy.)

AWS: Allows access based on date and time

This example shows how you might create an identity-based policy that allows access to actions based on date and time. This policy restricts access to actions that occur between April 1, 2020 and June 30, 2020 (UTC), inclusive. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

To learn about using multiple conditions within the `Condition` block of an IAM policy, see Multiple values in a condition.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "service-prefix:action-name",
            "Resource": "*",
            "Condition": {
                "DateGreaterThan": {"aws:CurrentTime": "2020-04-01T00:00:00Z"},
                "DateLessThan": {"aws:CurrentTime": "2020-06-30T23:59:59Z"}
            }
        }
    ]
}
```

Note

You cannot use a policy variable with the Date condition operator. To learn more see Condition element

AWS: Allows enabling and disabling AWS Regions

This example shows how you might create an identity-based policy that allows an administrator to enable and disable the Asia Pacific (Hong Kong) Region (ap-east-1). This policy defines permissions for programmatic and console access. This setting appears in the **Account settings** page in the AWS Management Console. This page includes sensitive account-level information that should be viewed and managed only by account administrators. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

Important

You cannot enable or disable regions that are enabled by default. You can only include regions that are *disabled* by default. For more information, see Managing AWS Regions in the *AWS General Reference*.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "EnableDisableHongKong",
            "Effect": "Allow",
            "Action": [
                "account:EnableRegion",
                "account:DisableRegion"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"account:TargetRegion": "ap-east-1"}
            }
        },
        {
            "Sid": "ViewConsole",
            "Effect": "Allow",
            "Action": [
                "account:ListRegions"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page

This example shows how you might create an identity-based policy that allows IAM users that are authenticated using multi-factor authentication (MFA) to manage their own credentials on the **Security credentials** page. This AWS Management Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, MFA devices, X.509 certificates, and SSH keys and Git credentials. This example policy includes the permissions required to view and edit all of the information on the page. It also requires the user to set up and authenticate using MFA before performing any other operations in AWS. To allow users to manage their own credentials without using MFA, see AWS: Allows IAM users to manage their own credentials on the Security credentials page.

To learn how users can access the **Security credentials** page, see How IAM users change their own password (console).

Note

- This example policy does not allow users to reset a password while signing in to the AWS Management Console for the first time. We recommend that you do not grant permissions to new users until after they sign in. For more information, see How do I securely create IAM users?. This also prevents users with an expired password from resetting their password during sign in. You can allow this by adding `iam:ChangePassword` and `iam:GetAccountPasswordPolicy` to the statement `DenyAllExceptListedIfNoMFA`. However, we do not recommend this because allowing users to change their password without MFA can be a security risk.
- If you intend to use this policy for programmatic access you must call `GetSessionToken` to authenticate with MFA. For more information, see Secure API access with MFA.

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view account-level information. These permissions must be in their own statement because they do not support or do not need to specify a resource ARN. Instead the permissions specify `"Resource" : "*"`. This statement includes the following actions that allow the user to view specific information:
  - `GetAccountPasswordPolicy` – View the account password requirements while changing their own IAM user password.
  - `ListVirtualMFADevices` – View details about a virtual MFA device that is enabled for the user.
- The `AllowManageOwnPasswords` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My security credentials** page.
- The `AllowManageOwnAccessKeys` statement allows the user to create, update, and delete their own access keys. The user can also retrieve information about when the specified access key was last used.
- The `AllowManageOwnSigningCertificates` statement allows the user to upload, update, and delete their own signing certificates.
- The `AllowManageOwnSSHPublicKeys` statement allows the user to upload, update, and delete their own SSH public keys for CodeCommit.

- The `AllowManageOwnGitCredentials` statement allows the user to create, update, and delete their own Git credentials for CodeCommit.
- The `AllowManageOwnVirtualMFADevice` statement allows the user to create their own virtual MFA device. The resource ARN in this statement allows the user to create an MFA device with any name, but the other statements in the policy only allow the user to attach the device to the currently signed-in user.
- The `AllowManageOwnUserMFA` statement allows the user to view or manage the virtual, U2F, or hardware MFA device for their own user. The resource ARN in this statement allows access to only the user's own IAM user. Users can't view or manage the MFA device for other users.
- The `DenyAllExceptListedIfNoMFA` statement denies access to every action in all AWS services, except a few listed actions, but ***only if*** the user is not signed in with MFA. The statement uses a combination of `"Deny"` and `"NotAction"` to explicitly deny access to every action that is not listed. The items listed are not denied or allowed by this statement. However, the actions are allowed by other statements in the policy. For more information about the logic for this statement, see NotAction with Deny. If the user is signed in with MFA, then the `Condition` test fails and this statement does not deny any actions. In this case, other policies or statements for the user determine the user's permissions.

  This statement ensures that when the user is not signed in with MFA that they can perform only the listed actions. In addition, they can perform the listed actions only if another statement or policy allows access to those actions. This does not allow a user to create a password at sign-in, because `iam:ChangePassword` action should not be allowed without MFA authorization.

  The `...IfExists` version of the `Bool` operator ensures that if the `aws:MultiFactorAuthPresent` key is missing, the condition returns true. This means that a user accessing an API with long-term credentials, such as an access key, is denied access to the non-IAM API operations.

This policy does not allow users to view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam:ListUsers` action to the `AllowViewAccountInfo` statement and the `DenyAllExceptListedIfNoMFA` statement. It also does not allow users to change their password on their own user page. To allow this, add the `iam:GetLoginProfile` and `iam:UpdateLoginProfile` actions to the `AllowManageOwnPasswords` statement. To also allow a user to change their password from their own user page without signing in using MFA, add the `iam:UpdateLoginProfile` action to the `DenyAllExceptListedIfNoMFA` statement.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccountPasswordPolicy",
                "iam:ListVirtualMFADevices"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswords",
            "Effect": "Allow",
            "Action": [
                "iam:ChangePassword",
                "iam:GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:CreateAccessKey",
                "iam:DeleteAccessKey",
                "iam:ListAccessKeys",
                "iam:UpdateAccessKey",
                "iam:GetAccessKeyLastUsed"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSigningCertificates",
            "Effect": "Allow",
            "Action": [
                "iam:DeleteSigningCertificate",
                "iam:ListSigningCertificates",
                "iam:UpdateSigningCertificate",
                "iam:UploadSigningCertificate"
```

```
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSSHPublicKeys",
            "Effect": "Allow",
            "Action": [
                "iam:DeleteSSHPublicKey",
                "iam:GetSSHPublicKey",
                "iam:ListSSHPublicKeys",
                "iam:UpdateSSHPublicKey",
                "iam:UploadSSHPublicKey"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnGitCredentials",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceSpecificCredential",
                "iam:DeleteServiceSpecificCredential",
                "iam:ListServiceSpecificCredentials",
                "iam:ResetServiceSpecificCredential",
                "iam:UpdateServiceSpecificCredential"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnVirtualMFADevice",
            "Effect": "Allow",
            "Action": [
                "iam:CreateVirtualMFADevice"
            ],
            "Resource": "arn:aws:iam::*:mfa/*"
        },
        {
            "Sid": "AllowManageOwnUserMFA",
            "Effect": "Allow",
            "Action": [
                "iam:DeactivateMFADevice",
                "iam:EnableMFADevice",
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
```

```
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "DenyAllExceptListedIfNoMFA",
            "Effect": "Deny",
            "NotAction": [
                "iam:CreateVirtualMFADevice",
                "iam:EnableMFADevice",
                "iam:GetUser",
                "iam:GetMFADevice",
                "iam:ListMFADevices",
                "iam:ListVirtualMFADevices",
                "iam:ResyncMFADevice",
                "sts:GetSessionToken"
            ],
            "Resource": "*",
            "Condition": {
                "BoolIfExists": {
                    "aws:MultiFactorAuthPresent": "false"
                }
            }
        }
    ]
}
```

AWS: Allows specific access using MFA within specific dates

This example shows how you might create an identity-based policy that uses multiple conditions, which are evaluated using a logical `AND`. It allows full access to the service named `SERVICE-NAME-1`, and access to the `ACTION-NAME-A` and `ACTION-NAME-B` actions in the service named `SERVICE-NAME-2`. These actions are allowed only when the user is authenticated using multifactor authentication (MFA). Access is restricted to actions that occur between July 1, 2017 and December 31, 2017 (UTC), inclusive. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

To learn about using multiple conditions within the `Condition` block of an IAM policy, see Multiple values in a condition.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "service-prefix-1:*",
            "service-prefix-2:action-name-a",
            "service-prefix-2:action-name-b"
        ],
        "Resource": "*",
        "Condition": {
            "Bool": {"aws:MultiFactorAuthPresent": true},
            "DateGreaterThan": {"aws:CurrentTime": "2017-07-01T00:00:00Z"},
            "DateLessThan": {"aws:CurrentTime": "2017-12-31T23:59:59Z"}
        }
    }
}
```

AWS: Allows IAM users to manage their own credentials on the Security credentials page

This example shows how you might create an identity-based policy that allows IAM users to manage all of their own credentials on the **Security credentials** page. This AWS Management Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, X.509 certificates, SSH keys, and Git credentials. This example policy includes the permissions required to view and edit all information on the page *except* the user's MFA device. To allow users to manage their own credentials with MFA, see AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page.

To learn how users can access the **Security credentials** page, see How IAM users change their own password (console).

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view account-level information. These permissions must be in their own statement because they do not support or do not need

to specify a resource ARN. Instead the permissions specify `"Resource" : "*"`. This statement includes the following actions that allow the user to view specific information:

- `GetAccountPasswordPolicy` – View the account password requirements while changing their own IAM user password.
- `GetAccountSummary` – View the account ID and the account canonical user ID.

- The `AllowManageOwnPasswords` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My security credentials** page.
- The `AllowManageOwnAccessKeys` statement allows the user to create, update, and delete their own access keys. The user can also retrieve information about when the specified access key was last used.
- The `AllowManageOwnSigningCertificates` statement allows the user to upload, update, and delete their own signing certificates.
- The `AllowManageOwnSSHPublicKeys` statement allows the user to upload, update, and delete their own SSH public keys for CodeCommit.
- The `AllowManageOwnGitCredentials` statement enables the user to create, update, and delete their own Git credentials for CodeCommit.

This policy does not allow users to view or manage their own MFA devices. They also cannot view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam:ListUsers` action to the `AllowViewAccountInfo` statement. It also does not allow users to change their password on their own user page. To allow this, add the `iam:CreateLoginProfile`, `iam:DeleteLoginProfile`, `iam:GetLoginProfile`, and `iam:UpdateLoginProfile` actions to the `AllowManageOwnPasswords` statement.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccountPasswordPolicy",
                "iam:GetAccountSummary"
```

```
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswords",
            "Effect": "Allow",
            "Action": [
                "iam:ChangePassword",
                "iam:GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:CreateAccessKey",
                "iam:DeleteAccessKey",
                "iam:ListAccessKeys",
                "iam:UpdateAccessKey",
                "iam:GetAccessKeyLastUsed"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSigningCertificates",
            "Effect": "Allow",
            "Action": [
                "iam:DeleteSigningCertificate",
                "iam:ListSigningCertificates",
                "iam:UpdateSigningCertificate",
                "iam:UploadSigningCertificate"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnSSHPublicKeys",
            "Effect": "Allow",
            "Action": [
                "iam:DeleteSSHPublicKey",
                "iam:GetSSHPublicKey",
                "iam:ListSSHPublicKeys",
                "iam:UpdateSSHPublicKey",
                "iam:UploadSSHPublicKey"
```

```
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnGitCredentials",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceSpecificCredential",
                "iam:DeleteServiceSpecificCredential",
                "iam:ListServiceSpecificCredentials",
                "iam:ResetServiceSpecificCredential",
                "iam:UpdateServiceSpecificCredential"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        }
    ]
}
```

AWS: Allows MFA-authenticated IAM users to manage their own MFA device on the Security credentials page

This example shows how you might create an identity-based policy that allows IAM users that are authenticated through multi-factor authentication (MFA) to manage their own MFA device on the **Security credentials** page. This AWS Management Console page displays account and user information, but the user can only view and edit their own MFA device. To allow users to manage all of their own credentials with MFA, see AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page.

> **Note**
>
> If an IAM user with this policy is not MFA-authenticated, this policy denies access to all AWS actions except those necessary to authenticate using MFA. To use the AWS CLI and AWS API, IAM users must first retrieve their MFA token using the AWS STS GetSessionToken operation and then use that token to authenticate the desired operation. Other policies, such as resource-based policies or other identity-based policies can allow actions in other services. This policy will deny that access if the IAM user is not MFA-authenticated.

To learn how users can access the **Security credentials** page, see How IAM users change their own password (console).

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view details about a virtual MFA device that is enabled for the user. This permission must be in its own statement because it does not support specifying a resource ARN. Instead you must specify `"Resource" : "*"`.
- The `AllowManageOwnVirtualMFADevice` statement allows the user to create their own virtual MFA device. The resource ARN in this statement allows the user to create an MFA device with any name, but the other statements in the policy only allow the user to attach the device to the currently signed-in user.
- The `AllowManageOwnUserMFA` statement allows the user to view or manage their own virtual, U2F, or hardware MFA device. The resource ARN in this statement allows access to only the user's own IAM user. Users can't view or manage the MFA device for other users.
- The `DenyAllExceptListedIfNoMFA` statement denies access to every action in all AWS services, except a few listed actions, but ***only if*** the user is not signed in with MFA. The statement uses a combination of `"Deny"` and `"NotAction"` to explicitly deny access to every action that is not listed. The items listed are not denied or allowed by this statement. However, the actions are allowed by other statements in the policy. For more information about the logic for this statement, see NotAction with Deny. If the user is signed in with MFA, then the `Condition` test fails and this statement does not deny any actions. In this case, other policies or statements for the user determine the user's permissions.

  This statement ensures that when the user is not signed in with MFA, they can perform only the listed actions. In addition, they can perform the listed actions only if another statement or policy allows access to those actions.

  The `...IfExists` version of the `Bool` operator ensures that if the `aws:MultiFactorAuthPresent` key is missing, the condition returns true. This means that a user accessing an API operation with long-term credentials, such as an access key, is denied access to the non-IAM API operations.

This policy does not allow users to view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam:ListUsers` action to the `AllowViewAccountInfo` statement and the `DenyAllExceptListedIfNoMFA` statement.

Warning

Do not add permission to delete an MFA device without MFA authentication. Users with this policy might attempt to assign themselves a virtual MFA device and receive an error that they are not authorized to perform `iam:DeleteVirtualMFADevice`. If this happens, **do not** add that permission to the `DenyAllExceptListedIfNoMFA` statement. Users that are not authenticated using MFA should never be allowed to delete their MFA device. Users might see this error if they previously began assigning a virtual MFA device to their user and cancelled the process. To resolve this issue, you or another administrator must delete the user's existing virtual MFA device using the AWS CLI or AWS API. For more information, see I am not authorized to perform: iam:DeleteVirtualMFADevice.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": "iam:ListVirtualMFADevices",
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnVirtualMFADevice",
            "Effect": "Allow",
            "Action": [
                "iam:CreateVirtualMFADevice"
            ],
            "Resource": "arn:aws:iam::*:mfa/*"
        },
        {
            "Sid": "AllowManageOwnUserMFA",
            "Effect": "Allow",
            "Action": [
                "iam:DeactivateMFADevice",
                "iam:EnableMFADevice",
                "iam:GetUser",
                "iam:GetMFADevice",
```

```
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "DenyAllExceptListedIfNoMFA",
            "Effect": "Deny",
            "NotAction": [
                "iam:CreateVirtualMFADevice",
                "iam:EnableMFADevice",
                "iam:GetUser",
                "iam:ListMFADevices",
                "iam:ListVirtualMFADevices",
                "iam:ResyncMFADevice",
                "sts:GetSessionToken"
            ],
            "Resource": "*",
            "Condition": {
                "BoolIfExists": {"aws:MultiFactorAuthPresent": "false"}
            }
        }
    ]
}
```

AWS: Allows IAM users to change their own console password on the Security credentials page

This example shows how you might create an identity-based policy that allows IAM users to change their own AWS Management Console password on the **Security credentials** page. This AWS Management Console page displays account and user information, but the user can only access their own password. To allow users to manage all of their own credentials with MFA, see AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page. To allow users to manage their own credentials without using MFA, see AWS: Allows IAM users to manage their own credentials on the Security credentials page.

To learn how users can access the **Security credentials** page, see How IAM users change their own password (console).

What does this policy do?

- The `ViewAccountPasswordRequirements` statement allows the user to view the account password requirements while changing their own IAM user password.
- The `ChangeOwnPassword` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My security credentials** page.

This policy does not allow users to view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam:ListUsers` action to the `ViewAccountPasswordRequirements` statement. It also does not allow users to change their password on their own user page. To allow this, add the `iam:GetLoginProfile` and `iam:UpdateLoginProfile` actions to the `ChangeOwnPassword` statement.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewAccountPasswordRequirements",
            "Effect": "Allow",
            "Action": "iam:GetAccountPasswordPolicy",
            "Resource": "*"
        },
        {
            "Sid": "ChangeOwnPassword",
            "Effect": "Allow",
            "Action": [
                "iam:GetUser",
                "iam:ChangePassword"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        }
    ]
}
```

AWS: Allows IAM users to manage their own password, access keys, and SSH public keys on the Security credentials page

This example shows how you might create an identity-based policy that allows IAM users to manage their own password, access keys, and X.509 certificates on the **Security credentials** page. This AWS Management Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, MFA devices, X.509 certificates, SSH keys, and Git credentials. This example policy includes the permissions that are required to view and edit only their password, access keys, and X.509 certificate. To allow users to manage all of their own credentials with MFA, see AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page. To allow users to manage their own credentials without using MFA, see AWS: Allows IAM users to manage their own credentials on the Security credentials page.

To learn how users can access the **Security credentials** page, see How IAM users change their own password (console).

What does this policy do?

- The `AllowViewAccountInfo` statement allows the user to view account-level information. These permissions must be in their own statement because they do not support or do not need to specify a resource ARN. Instead the permissions specify `"Resource" : "*"`. This statement includes the following actions that allow the user to view specific information:
  - `GetAccountPasswordPolicy` – View the account password requirements while changing their own IAM user password.
  - `GetAccountSummary` – View the account ID and the account canonical user ID.
- The `AllowManageOwnPasswords` statement allows the user to change their own password. This statement also includes the `GetUser` action, which is required to view most of the information on the **My security credentials** page.
- The `AllowManageOwnAccessKeys` statement allows the user to create, update, and delete their own access keys. The user can also retrieve information about when the specified access key was last used.
- The `AllowManageOwnSSHPublicKeys` statement allows the user to upload, update, and delete their own SSH public keys for CodeCommit.

This policy does not allow users to view or manage their own MFA devices. They also cannot view the **Users** page in the IAM console or use that page to access their own user information. To allow this, add the `iam:ListUsers` action to the `AllowViewAccountInfo` statement. It also does not allow users to change their password on their own user page. To allow this, add the `iam:GetLoginProfile` and `iam:UpdateLoginProfile` actions to the `AllowManageOwnPasswords` statement.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewAccountInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccountPasswordPolicy",
                "iam:GetAccountSummary"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswords",
            "Effect": "Allow",
            "Action": [
                "iam:ChangePassword",
                "iam:GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowManageOwnAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:CreateAccessKey",
                "iam:DeleteAccessKey",
                "iam:ListAccessKeys",
                "iam:UpdateAccessKey",
                "iam:GetAccessKeyLastUsed"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
```

```
        {
            "Sid": "AllowManageOwnSSHPublicKeys",
            "Effect": "Allow",
            "Action": [
                "iam:DeleteSSHPublicKey",
                "iam:GetSSHPublicKey",
                "iam:ListSSHPublicKeys",
                "iam:UpdateSSHPublicKey",
                "iam:UploadSSHPublicKey"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        }
    ]
}
```

AWS: Denies access to AWS based on the requested Region

This example shows how you might create an identity-based policy that denies access to any actions outside the Regions specified using the `aws:RequestedRegion` condition key, except for actions in the services specified using `NotAction`. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

This policy uses the `NotAction` element with the `Deny` effect, which explicitly denies access to all of the actions *not* listed in the statement. Actions in the CloudFront, IAM, Route 53, and Support services should not be denied because these are popular AWS global services with a single endpoint that is physically located in the `us-east-1` Region. Because all requests to these services are made to the `us-east-1` Region, the requests would be denied without the `NotAction` element. Edit this element to include actions for other AWS global services that you use, such as `budgets`, `globalaccelerator`, `importexport`, `organizations`, or `waf`. Some other global services, such as Amazon Q Developer in chat applications and AWS Device Farm, are global services with endpoints that are physically located in the `us-west-2` region. To learn about all of the services that have a single global endpoint, see AWS Regions and Endpoints in the *AWS General Reference*. For more information about using the `NotAction` element with the `Deny` effect, see IAM JSON policy elements: NotAction.

Important

This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DenyAllOutsideRequestedRegions",
            "Effect": "Deny",
            "NotAction": [
                "cloudfront:*",
                "iam:*",
                "organizations:*",
                "route53:*",
                "support:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestedRegion": [
                        "eu-central-1",
                        "eu-west-1",
                        "eu-west-2",
                        "eu-west-3"
                    ]
                }
            }
        }
    ]
}
```

AWS: Denies access to AWS based on the source IP

This example shows how you might create an identity-based policy that denies access to all AWS actions in the account when the request comes from *principals* outside the specified IP range.

The policy is useful when the IP addresses for your company are within the specified ranges. In this example, the request will be denied unless it originates from the CIDR range 192.0.2.0/24 or 203.0.113.0/24. The policy does not deny requests made by AWS services using Forward access sessions as the original requester's IP address is preserved.

Be careful using negative conditions in the same policy statement as `"Effect": "Deny"`. When you do, the actions specified in the policy statement are explicitly denied in all conditions *except* for the ones specified.

> **Important**
>
> This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

When other policies allow actions, principals can make requests from within the IP address range. An AWS service can also make requests using the principal's credentials. When a principal makes a request from outside the IP range, the request is denied.

For more information about using the `aws:SourceIp` condition key, including information about when `aws:SourceIp` may not work in your policy, see AWS global condition context keys.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "NotIpAddress": {
                "aws:SourceIp": [
                    "192.0.2.0/24",
                    "203.0.113.0/24"
                ]
            }
        }
    }
}
```

AWS: Deny access to Amazon S3 resources outside your account except AWS Data Exchange

This example shows how you might create an identity-based policy that denies access to all resources in AWS that don't belong to your account, except for the resources that AWS Data Exchange requires for normal operation. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

You can create a similar policy to restrict access to resources within an organization or an organizational unit, while accounting for AWS Data Exchange owned resources by using the condition keys `aws:ResourceOrgPaths` and `aws:ResourceOrgID`.

If you use AWS Data Exchange in your environment, the service creates and interacts with resources such as Amazon S3 buckets owned by the service account. For example, AWS Data Exchange sends requests to Amazon S3 buckets owned by the AWS Data Exchange service on behalf of the IAM principal (user or role) invoking the AWS Data Exchange APIs. In that case, using `aws:ResourceAccount`, `aws:ResourceOrgPaths`, or `aws:ResourceOrgID` in a policy, without accounting for AWS Data Exchange owned resources, denies access to the buckets owned by the service account.

- The statement, `DenyAllAwsResourcesOutsideAccountExceptS3`, uses the `NotAction` element with the Deny effect which explicitly denies access to every action not listed in the statement that also do not belong to the listed account. The `NotAction` element indicates the exceptions to this statement. These actions are the exception to this statement because if the actions are performed on resources created by AWS Data Exchange, the policy denies them.
- The statement, `DenyAllS3ResourcesOutsideAccountExceptDataExchange`, uses a combination of the `ResourceAccount` and `CalledVia` conditions to deny access to the three Amazon S3 actions excluded in the previous statement. The statement denies the actions if resources do not belong in the listed account and if the calling service is not AWS Data Exchange. The statement does not deny the actions if either the resource belongs to the listed account or the listed service principal, `dataexchange.amazonaws.com`, performs the operations.

> **Important**
>
> This policy does not allow any actions. It uses the `Deny` effect which explicitly denies access to all of the resources listed in the statement that do not belong to the listed account. Use this policy in combination with other policies that allow access to specific resources.

The following example shows how you can configure the policy to allow access to the required Amazon S3 buckets.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAllAwsReourcesOutsideAccountExceptAmazonS3",
      "Effect": "Deny",
      "NotAction": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "111122223333"
          ]
        }
      }
    },
    {
      "Sid": "DenyAllS3ResourcesOutsideAccountExceptDataExchange",
      "Effect": "Deny",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "*",
```

```
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "111122223333"
          ]
        },
        "ForAllValues:StringNotEquals": {
          "aws:CalledVia": [
            "dataexchange.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS Data Pipeline: Denies access to DataPipeline pipelines that a user did not create

This example shows how you might create an identity-based policy that denies access to pipelines that a user did not create. If the value of the `PipelineCreator` field matches the IAM user name, then the specified actions are not denied. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI.

> **Important**
>
> This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExplicitDenyIfNotTheOwner",
            "Effect": "Deny",
            "Action": [
```

```
                "datapipeline:ActivatePipeline",
                "datapipeline:AddTags",
                "datapipeline:DeactivatePipeline",
                "datapipeline:DeletePipeline",
                "datapipeline:DescribeObjects",
                "datapipeline:EvaluateExpression",
                "datapipeline:GetPipelineDefinition",
                "datapipeline:PollForTask",
                "datapipeline:PutPipelineDefinition",
                "datapipeline:QueryObjects",
                "datapipeline:RemoveTags",
                "datapipeline:ReportTaskProgress",
                "datapipeline:ReportTaskRunnerHeartbeat",
                "datapipeline:SetStatus",
                "datapipeline:SetTaskStatus",
                "datapipeline:ValidatePipelineDefinition"
            ],
            "Resource": ["*"],
            "Condition": {
                "StringNotEquals": {"datapipeline:PipelineCreator":
 "${aws:userid}"}
            }
        }
    ]
}
```

Amazon DynamoDB: Allows access to a specific table

This example shows how you might create an identity-based policy that allows full access to the `MyTable` DynamoDB table. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

> **Important**
>
> This policy allows all actions that can be performed on a DynamoDB table. To review these actions, see DynamoDB API Permissions: Actions, Resources, and Conditions Reference in the *Amazon DynamoDB Developer Guide*. You could provide the same permissions by listing each individual action. However, if you use the wildcard (*) in the `Action` element, such as

"dynamodb:List*", then you don't have to update your policy if DynamoDB adds a new List action.

This policy allows actions only on DynamoDB tables that exist with the specified name. To allow your users Read access to everything in DynamoDB, you can also attach the AmazonDynamoDBReadOnlyAccess AWS managed policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListAndDescribe",
            "Effect": "Allow",
            "Action": [
                "dynamodb:List*",
                "dynamodb:DescribeReservedCapacity*",
                "dynamodb:DescribeLimits",
                "dynamodb:DescribeTimeToLive"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SpecificTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGet*",
                "dynamodb:DescribeStream",
                "dynamodb:DescribeTable",
                "dynamodb:Get*",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWrite*",
                "dynamodb:CreateTable",
                "dynamodb:Delete*",
                "dynamodb:Update*",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/MyTable"
        }
```

```
    ]
}
```

Amazon DynamoDB: Allows access to specific attributes

This example shows how you might create an identity-based policy that allows access to the specific DynamoDB attributes. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The `dynamodb:Select` requirement prevents the API action from returning any attributes that aren't allowed, such as from an index projection. To learn more about DynamoDB condition keys, see Specifying Conditions: Using Condition Keys in the *Amazon DynamoDB Developer Guide*. To learn about using multiple conditions or multiple condition keys within the `Condition` block of an IAM policy, see Multiple values in a condition.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": ["arn:aws:dynamodb:*:*:table/table-name"],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:Attributes": [
                        "column-name-1",
                        "column-name-2",
                        "column-name-3"
```

```
                    ]
                },
                "StringEqualsIfExists": {"dynamodb:Select":
 "SPECIFIC_ATTRIBUTES"}
            }
        }
    ]
}
```

Amazon DynamoDB: Allows item-level access to DynamoDB based on an Amazon Cognito ID

This example shows how you might create an identity-based policy that allows item-level access to the `MyTable` DynamoDB table based on an Amazon Cognito identity pool user ID. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

To use this policy, you must structure your DynamoDB table so the Amazon Cognito identity pool user ID is the partition key. For more information, see Creating a Table in the *Amazon DynamoDB Developer Guide*.

To learn more about DynamoDB condition keys, see Specifying Conditions: Using Condition Keys in the *Amazon DynamoDB Developer Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": ["arn:aws:dynamodb:*:*:table/MyTable"],
```

```
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": ["${cognito-
identity.amazonaws.com:sub}"]
                }
            }
        }
    ]
}
```

Amazon EC2: Attach or detach Amazon EBS volumes to EC2 instances based on tags

This example shows how you might create an identity-based policy that allows EBS volume owners to attach or detach their EBS volumes defined using the tag `VolumeUser` to EC2 instances that are tagged as development instances (`Department=Development`). This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

For more information about creating IAM policies to control access to Amazon EC2 resources, see Controlling Access to Amazon EC2 Resources in the *Amazon EC2 User Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachVolume",
                "ec2:DetachVolume"
            ],
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Department": "Development"}
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:AttachVolume",
                "ec2:DetachVolume"
            ],
            "Resource": "arn:aws:ec2:*:*:volume/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/VolumeUser": "${aws:username}"}
            }
        }
    ]
}
```

Amazon EC2: Allows launching EC2 instances in a specific subnet, programmatically and in the console

This example shows how you might create an identity-based policy that allows listing information for all EC2 objects and launching EC2 instances in a specific subnet. This policy defines permissions for programmatic and console access. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:GetConsole*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": [
                "arn:aws:ec2:*:*:subnet/subnet-subnet-id",
```

```
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*::image/ami-*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:security-group/*"
            ]
        }
    ]
}
```

Amazon EC2: Allows managing EC2 security groups with a specific tag key-value pair programmatically and in the console

This example shows how you might create an identity-based policy that grants users permission to take certain actions for security groups that have the same tag. This policy grants permissions to view security groups in the Amazon EC2 console, add and remove inbound and outbound rules, and list and modify rule descriptions for existing security groups with the tag `Department=Test`. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": [
         "ec2:DescribeSecurityGroups",
         "ec2:DescribeSecurityGroupRules",
         "ec2:DescribeTags"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
         "ec2:AuthorizeSecurityGroupIngress",
         "ec2:RevokeSecurityGroupIngress",
```

```
            "ec2:AuthorizeSecurityGroupEgress",
            "ec2:RevokeSecurityGroupEgress",
            "ec2:ModifySecurityGroupRules",
            "ec2:UpdateSecurityGroupRuleDescriptionsIngress",
            "ec2:UpdateSecurityGroupRuleDescriptionsEgress"
        ],
        "Resource": [
            "arn:aws:ec2:us-east-1:111122223333:security-group/*"
        ],
        "Condition": {
            "StringEquals": {
                "aws:ResourceTag/Department": "Test"
            }
        }
      },
      {
        "Effect": "Allow",
        "Action": [
            "ec2:ModifySecurityGroupRules"
        ],
        "Resource": [
            "arn:aws:ec2:us-east-1:111122223333:security-group-rule/*"
        ]
      }
    ]
}
```

Amazon EC2: Allows starting or stopping EC2 instances a user has tagged, programmatically and in the console

This example shows how you might create an identity-based policy that allows an IAM user to start or stop EC2 instances, but only if the instance tag `Owner` has the value of that user's user name. This policy defines permissions for programmatic and console access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "ec2:StartInstances",
                "ec2:StopInstances"
            ],
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:DescribeInstances",
            "Resource": "*"
        }
    ]
}
```

EC2: Start or stop instances based on tags

This example shows how you might create an identity-based policy that allows starting or stopping instances with the tag key–value pair `Project = DataAnalytics`, but only by principals with the tag key–value pair `Department = Data`. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The condition in the policy returns true if both parts of the condition are true. The instance must have the `Project=DataAnalytics` tag. In addition, the IAM principal (user or role) making the request must have the `Department=Data` tag.

> **Note**
>
> As a best practice, attach policies with the `aws:PrincipalTag` condition key to IAM groups, for the case where some users might have the specified tag and some might not.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "StartStopIfTags",
            "Effect": "Allow",
            "Action": [
                "ec2:StartInstances",
                "ec2:StopInstances"
            ],
            "Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Project": "DataAnalytics",
                    "aws:PrincipalTag/Department": "Data"
                }
            }
        }
    ]
}
```

EC2: Start or stop instances based on matching principal and resource tags

This example shows how you might create an identity-based policy that allows a principal to start or stop an Amazon EC2 instance when the instance's resource tag and the principal's tag have the same value for the tag key `CostCenter`. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

> **Note**
>
> As a best practice, attach policies with the `aws:PrincipalTag` condition key to IAM groups, for the case where some users might have the specified tag and some might not.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "ec2:startInstances",
            "ec2:stopInstances"
        ],
        "Resource": "*",
        "Condition": {"StringEquals":
            {"aws:ResourceTag/CostCenter": "${aws:PrincipalTag/CostCenter}"}}
    }
}
```

Amazon EC2: Allows full EC2 access within a specific Region, programmatically and in the console

This example shows how you might create an identity-based policy that allows full EC2 access within a specific Region. This policy defines permissions for programmatic and console access. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy. For a list of Region codes, see Available Regions in the *Amazon EC2 User Guide*.

Alternatively, you can use the global condition key `aws:RequestedRegion`, which is supported by all Amazon EC2 API actions. For more information, see Example: Restricting access to a specific Region in the *Amazon EC2 User Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": "ec2:*",
            "Resource": "*",
            "Effect": "Allow",
            "Condition": {
```

```
                "StringEquals": {
                    "ec2:Region": "us-east-2"
                }
            }
        }
    ]
}
```

Amazon EC2: Allows starting or stopping an EC2 instance and modifying a security group, programmatically and in the console

This example shows how you might create an identity-based policy that allows starting or stopping a specific EC2 instance and modifying a specific security group. This policy defines permissions for programmatic and console access. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSecurityGroupReferences",
        "ec2:DescribeStaleSecurityGroups"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ec2:AuthorizeSecurityGroupEgress",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:RevokeSecurityGroupEgress",
        "ec2:RevokeSecurityGroupIngress",
        "ec2:StartInstances",
        "ec2:StopInstances"
```

```
      ],
      "Resource": [
        "arn:aws:ec2:*:*:instance/i-instance-id",
        "arn:aws:ec2:*:*:security-group/sg-security-group-id"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Amazon EC2: Requires MFA (GetSessionToken) for specific EC2 operations

This example shows how you might create an identity-based policy that allows full access to all AWS API operations in Amazon EC2. However, it explicitly denies access to `StopInstances` and `TerminateInstances` API operations if the user is not authenticated using multi-factor authentication (MFA). To do this programmatically, the user must include optional `SerialNumber` and `TokenCode` values while calling the `GetSessionToken` operation. This operation returns temporary credentials that were authenticated using MFA. To learn more about GetSessionToken, see Requesting credentials for users in untrusted environments.

What does this policy do?

- The `AllowAllActionsForEC2` statement allows all Amazon EC2 actions.
- The `DenyStopAndTerminateWhenMFAIsNotPresent` statement denies the `StopInstances` and `TerminateInstances` actions when the MFA context is missing. This means that the actions are denied when the multi-factor authentication context is missing (meaning MFA was not used). A deny overrides the allow.

> **Note**
>
> The condition check for `MultiFactorAuthPresent` in the `Deny` statement should not be a `{"Bool":{"aws:MultiFactorAuthPresent":false}}` because that key is not present and cannot be evaluated when MFA is not used. So instead, use the `BoolIfExists` check to see whether the key is present before checking the value. For more information, see ...IfExists condition operators.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllActionsForEC2",
            "Effect": "Allow",
            "Action": "ec2:*",
            "Resource": "*"
        },
        {
            "Sid": "DenyStopAndTerminateWhenMFAIsNotPresent",
            "Effect": "Deny",
            "Action": [
                "ec2:StopInstances",
                "ec2:TerminateInstances"
            ],
            "Resource": "*",
            "Condition": {
                "BoolIfExists": {"aws:MultiFactorAuthPresent": false}
            }
        }
    ]
}
```

Amazon EC2: Limits terminating EC2 instances to an IP address range

This example shows how you might create an identity-based policy that limits EC2 instances by allowing the action, but explicitly denying access when the request comes from outside the specified IP range. The policy is useful when the IP addresses for your company are within the specified ranges. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

If this policy is used in combination with other policies that allow the `ec2:TerminateInstances` action (such as the AmazonEC2FullAccess AWS managed policy), then access is denied. This is because an explicit deny statement takes precedence over allow statements. For more information, see How AWS enforcement code logic evaluates requests to allow or deny access.

Important

The `aws:SourceIp` condition key denies access to an AWS service, such as AWS CloudFormation, that makes calls on your behalf. For more information about using the `aws:SourceIp` condition key, see AWS global condition context keys.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ec2:TerminateInstances"],
            "Resource": ["*"]
        },
        {
            "Effect": "Deny",
            "Action": ["ec2:TerminateInstances"],
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            },
            "Resource": ["*"]
        }
    ]
}
```

IAM: Access the policy simulator API

This example shows how you might create an identity-based policy that allows using the policy simulator API for policies attached to a user, group, or role in the current AWS account. This policy also allows access to simulate less sensitive policies passed to the API as strings. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:GetContextKeysForCustomPolicy",
                "iam:GetContextKeysForPrincipalPolicy",
                "iam:SimulateCustomPolicy",
                "iam:SimulatePrincipalPolicy"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Note

To allow a user to access the policy simulator console to simulate policies attached to a user, group, or role in the current AWS account, see IAM: Access the policy simulator console.

IAM: Access the policy simulator console

This example shows how you might create an identity-based policy that allows using the policy simulator console for policies attached to a user, group, or role in the current AWS account.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "access-analyzer:ValidatePolicy",
                "iam:GetContextKeysForCustomPolicy",
```

```
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:GetUser",
                "iam:GetUserPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListAttachedRolePolicies",
                "iam:ListAttachedUserPolicies",
                "iam:ListGroups",
                "iam:ListGroupPolicies",
                "iam:ListGroupsForUser",
                "iam:ListPolicies",
                "iam:ListRolePolicies",
                "iam:ListRoles",
                "iam:ListUserPolicies",
                "iam:ListUsers",
                "iam:SimulateCustomPolicy",
                "iam:SimulatePrincipalPolicy"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

IAM: Assume roles that have a specific tag

This example shows how you might create an identity-based policy that allows an IAM user to assume roles with the tag key-value pair `Project = ExampleCorpABC`. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

If a role with this tag exists in the same account as the user, then the user can assume that role. If a role with this tag exists in an account other than the user's, it requires additional permissions. The cross-account role's trust policy must also allow the user or all members of the user's account to assume the role. For information about using roles for cross-account access, see Access for an IAM user in another AWS account that you own.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeTaggedRole",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {"iam:ResourceTag/Project": "ExampleCorpABC"}
            }
        }
    ]
}
```

IAM: Allows and denies access to multiple services programmatically and in the console

This example shows how you might create an identity-based policy that allows full access to several services and limited self-managing access in IAM. It also denies access to the Amazon S3 `logs` bucket or the Amazon EC2 `i-1234567890abcdef0` instance. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

> **Warning**
>
> This policy allows full access to every action and resource in multiple services. This policy should be applied only to trusted administrators.

You can use this policy as a permissions boundary to define the maximum permissions that an identity-based policy can grant to an IAM user. For more information, see Delegating responsibility to others using permissions boundaries. When the policy is used as a permissions boundary for a user, the statements define the following boundaries:

- The `AllowServices` statement allows full access to the specified AWS services. This means that the user's actions in these services are limited only by the permissions policies that are attached to the user.
- The `AllowIAMConsoleForCredentials` statement allows access to list all IAM users. This access is necessary to navigate the **Users** page in the AWS Management Console. It also allows viewing the password requirements for the account, which is necessary for the user to change their own password.
- The `AllowManageOwnPasswordAndAccessKeys` statement allows the users manage only their own console password and programmatic access keys. This is important because if another policy gives a user full IAM access, that user could then change their own or other users' permissions. This statement prevents that from happening.
- The `DenyS3Logs` statement explicitly denies access to the `logs` bucket. This policy enforces company restrictions on the user.
- The `DenyEC2Production` statement explicitly denies access to the `i-1234567890abcdef0` instance.

This policy does not allow access to other services or actions. When the policy is used as a permissions boundary on a user, even if other policies attached to the user allow those actions, AWS denies the request.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowServices",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMConsoleForCredentials",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:ListUsers",
                "iam:GetAccountPasswordPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswordAndAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:*AccessKey*",
                "iam:ChangePassword",
                "iam:GetUser",
                "iam:*LoginProfile*"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::logs",
                "arn:aws:s3:::logs/*"
            ]
        },
        {
            "Sid": "DenyEC2Production",
            "Effect": "Deny",
            "Action": "ec2:*",
            "Resource": "arn:aws:ec2:*:*:instance/i-1234567890abcdef0"
        }
    ]
}
```

IAM: Add a specific tag to a user with a specific tag

This example shows how you might create an identity-based policy that allows adding the tag key `Department` with the tag values `Marketing`, `Development`, or `QualityAssurance` to an IAM user. That user must already include the tag key–value pair `JobFunction = Manager`. You can use this policy to require that a manager belong to only one of three departments. This policy defines permissions for programmatic and console access. To use this policy, replace the

italicized placeholder text in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The `ListTagsForAllUsers` statement allows the viewing of tags for all users in your account.

The first condition in the `TagManagerWithSpecificDepartment` statement uses the `StringEquals` condition operator. The condition returns true if both parts of the condition are true. The user to be tagged must already have the `JobFunction=Manager` tag. The request must include the `Department` tag key with one of the listed tag values.

The second condition uses the `ForAllValues:StringEquals` condition operator. The condition returns true if all of the tag keys in the request match the key in the policy. This means that the only tag key in the request must be `Department`. For more information about using `ForAllValues`, see Set operators for multivalued context keys.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListTagsForAllUsers",
            "Effect": "Allow",
            "Action": [
                "iam:ListUserTags",
                "iam:ListUsers"
            ],
            "Resource": "*"
        },
        {
            "Sid": "TagManagerWithSpecificDepartment",
            "Effect": "Allow",
            "Action": "iam:TagUser",
            "Resource": "*",
            "Condition": {"StringEquals": {
                "iam:ResourceTag/JobFunction": "Manager",
                "aws:RequestTag/Department": [
                    "Marketing",
                    "Development",
                    "QualityAssurance"
                    ]
                },
```

```
                "ForAllValues:StringEquals": {"aws:TagKeys": "Department"}
            }
        }
    ]
}
```

IAM: Add a specific tag with specific values

This example shows how you might create an identity-based policy that allows adding only the tag key `CostCenter` and either the tag value `A-123` or the tag value `B-456` to any IAM user or role. You can use this policy to limit tagging to a specific tag key and set of tag values. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The `ConsoleDisplay` statement allows the viewing of tags for all users and roles in your account.

The first condition in the `AddTag` statement uses the `StringEquals` condition operator. The condition returns true if the request includes the `CostCenter` tag key with one of the listed tag values.

The second condition uses the `ForAllValues:StringEquals` condition operator. The condition returns true if all of the tag keys in the request match the key in the policy. This means that the only tag key in the request must be `CostCenter`. For more information about using `ForAllValues`, see Set operators for multivalued context keys.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ConsoleDisplay",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:GetUser",
                "iam:ListRoles",
                "iam:ListRoleTags",
                "iam:ListUsers",
```

```
                "iam:ListUserTags"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AddTag",
            "Effect": "Allow",
            "Action": [
                "iam:TagUser",
                "iam:TagRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CostCenter": [
                        "A-123",
                        "B-456"
                    ]
                },
                "ForAllValues:StringEquals": {"aws:TagKeys": "CostCenter"}
            }
        }
    ]
}
```

IAM: Create new users only with specific tags

This example shows how you might create an identity-based policy that allows the creation of IAM users but only with one or both of the `Department` and `JobFunction` tag keys. The `Department` tag key must have either the `Development` or `QualityAssurance` tag value. The `JobFunction` tag key must have the `Employee` tag value. You can use this policy to require that new users have a specific job function and department. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The first condition in the statement uses the `StringEqualsIfExists` condition operator. If a tag with the `Department` or `JobFunction` key is present in the request, then the tag must have the specified value. If neither key is present, then this condition is evaluated as true. The only way that the condition evaluates as false is if one of the specified condition keys is present in the request,

but has a different value than those allowed. For more information about using `IfExists`, see ...IfExists condition operators.

The second condition uses the `ForAllValues:StringEquals` condition operator. The condition returns true if there's a match between every one of the specified tag keys specified in the request, and at least one value in the policy. This means that all of the tags in the request must be in this list. However, the request can include only one of the tags in the list. For example, you can create an IAM user with only the `Department=QualityAssurance` tag. However, you cannot create an IAM user with the `JobFunction=Employee` tag and the `Project=core` tag. For more information about using `ForAllValues`, see Set operators for multivalued context keys.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "TagUsersWithOnlyTheseTags",
            "Effect": "Allow",
            "Action": [
                "iam:CreateUser",
                "iam:TagUser"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "aws:RequestTag/Department": [
                        "Development",
                        "QualityAssurance"
                    ],
                    "aws:RequestTag/JobFunction": "Employee"
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "Department",
                        "JobFunction"
                    ]
                }
            }
        }
    ]
```

```
}
```

IAM: Generate and retrieve IAM credential reports

This example shows how you might create an identity-based policy that allows users to generate and download a report that lists all IAM users in their AWS account. The report includes the status of the users' credentials, including passwords, access keys, MFA devices, and signing certificates. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI.

For more information about credential reports, see Generate credential reports for your AWS account.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GenerateCredentialReport",
            "iam:GetCredentialReport"
        ],
        "Resource": "*"
    }
}
```

IAM: Allows managing a group's membership programmatically and in the console

This example shows how you might create an identity-based policy that allows updating the membership of the group called `MarketingTeam`. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

What does this policy do?

- The `ViewGroups` statement allows the user to list all the users and groups in the AWS Management Console. It also allows the user to view basic information about the users in the account. These permissions must be in their own statement because they do not support or do not need to specify a resource ARN. Instead the permissions specify `"Resource" : "*"`.
- The `ViewEditThisGroup` statement allows the user to view information about the `MarketingTeam` group, and to add and remove users from that group.

This policy does not allow the user to view or edit the permissions of the users or the `MarketingTeam` group.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewGroups",
            "Effect": "Allow",
            "Action": [
                "iam:ListGroups",
                "iam:ListUsers",
                "iam:GetUser",
                "iam:ListGroupsForUser"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ViewEditThisGroup",
            "Effect": "Allow",
            "Action": [
                "iam:AddUserToGroup",
                "iam:RemoveUserFromGroup",
                "iam:GetGroup"
            ],
            "Resource": "arn:aws:iam::*:group/MarketingTeam"
        }
    ]
}
```

IAM: Manage a specific tag

This example shows how you might create an identity-based policy that allows adding and removing the IAM tag with the tag key `Department` from IAM entities (users and roles). This policy does not limit the value of the `Department` tag. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:TagUser",
            "iam:TagRole",
            "iam:UntagUser",
            "iam:UntagRole"

        ],
        "Resource": "*",
        "Condition": {"ForAllValues:StringEquals": {"aws:TagKeys": "Department"}}
    }
}
```

IAM: Pass an IAM role to a specific AWS service

This example shows how you might create an identity-based policy that allows passing any IAM service role to the Amazon CloudWatch service. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

A service role is an IAM role that specifies an AWS service as the principal that can assume the role. This allows the service to assume the role and access resources in other services on your behalf. To allow Amazon CloudWatch to assume the role that you pass, you must specify the `cloudwatch.amazonaws.com` service principal as the principal in the trust policy of your role.

The service principal is defined by the service. To learn the service principal for a service, see the documentation for that service. For some services, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service. Search for `amazonaws.com` to view the service principal.

To learn more about passing a service role to a service, see Grant a user permissions to pass a role to an AWS service.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {"iam:PassedToService":
 "cloudwatch.amazonaws.com"}
            }
        }
    ]
}
```

IAM: Allows read-only access to the IAM console without reporting

This example shows how you might create an identity-based policy that allows IAM users to perform any IAM action that begins with the string `Get` or `List`. As users work with the console, the console makes requests to IAM to list groups, users, roles, and policies, and to generate reports about those resources.

The asterisk acts as a wildcard. When you use `iam:Get*` in a policy, the resulting permissions include all IAM actions that begin with `Get`, such as `GetUser` and `GetRole`. Wildcards are useful if new types of entities are added to IAM in the future. In that case, the permissions granted by the policy automatically allow the user to list and get the details about those new entities.

This policy cannot be used to generate reports or service last accessed details. For a different policy that allows this, see IAM: Allows read-only access to the IAM console.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:Get*",
            "iam:List*"
        ],
        "Resource": "*"
    }
}
```

IAM: Allows read-only access to the IAM console

This example shows how you might create an identity-based policy that allows IAM users to perform any IAM action that begins with the string `Get`, `List`, or `Generate`. As users work with the IAM console, the console makes requests to list groups, users, roles, and policies, and to generate reports about those resources.

The asterisk acts as a wildcard. When you use `iam:Get*` in a policy, the resulting permissions include all IAM actions that begin with `Get`, such as `GetUser` and `GetRole`. Using a wildcard is beneficial, especially if new types of entities are added to IAM in the future. In that case, the permissions granted by the policy automatically allow the user to list and get the details about those new entities.

Use this policy for console access that includes permissions to generate reports or service last accessed details. For a different policy that does not allow generating actions, see IAM: Allows read-only access to the IAM console without reporting.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:Get*",
            "iam:List*",
            "iam:Generate*"
        ],
        "Resource": "*"
    }
}
```

IAM: Allows specific IAM users to manage a group programmatically and in the console

This example shows how you might create an identity-based policy that allows specific IAM users to manage the `AllUsers` group. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

What does this policy do?

- The `AllowAllUsersToListAllGroups` statement allows listing all groups. This is necessary for console access. This permission must be in its own statement because it does not support a resource ARN. Instead the permissions specify `"Resource" : "*"`.
- The `AllowAllUsersToViewAndManageThisGroup` statement allows all group actions that can be performed on the group resource type. It does not allow the `ListGroupsForUser` action, which can be performed on a user resource type and not a group resource type. For more information about the resource types that you can specify for an IAM action, see Actions, Resources, and Condition Keys for AWS Identity and Access Management.
- The `LimitGroupManagementAccessToSpecificUsers` statement denies users with the specified names access to write and permissions management group actions. When a user specified in the policy attempts to make changes to the group, this statement does not deny the request. That request is allowed by the `AllowAllUsersToViewAndManageThisGroup` statement. If other users attempt to perform these operations, the request is denied. You can view the IAM actions that are defined with the **Write** or **Permissions management** access levels while creating this policy in the IAM console. To do this, switch from the **JSON** tab to the **Visual editor** tab. For more information about access levels, see Actions, Resources, and Condition Keys for AWS Identity and Access Management.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllUsersToListAllGroups",
            "Effect": "Allow",
            "Action": "iam:ListGroups",
            "Resource": "*"
        },
        {
            "Sid": "AllowAllUsersToViewAndManageThisGroup",
            "Effect": "Allow",
            "Action": "iam:*Group*",
            "Resource": "arn:aws:iam::*:group/AllUsers"
        },
        {
            "Sid": "LimitGroupManagementAccessToSpecificUsers",
            "Effect": "Deny",
            "Action": [
                "iam:AddUserToGroup",
                "iam:CreateGroup",
                "iam:RemoveUserFromGroup",
                "iam:DeleteGroup",
                "iam:AttachGroupPolicy",
                "iam:UpdateGroup",
                "iam:DetachGroupPolicy",
                "iam:DeleteGroupPolicy",
                "iam:PutGroupPolicy"
            ],
            "Resource": "arn:aws:iam::*:group/AllUsers",
            "Condition": {
                "StringNotEquals": {
                    "aws:username": [
                        "srodriguez",
                        "mjackson",
                        "adesai"
                    ]
                }
            }
        }
    ]
```

```
}
```

IAM: Allows setting the account password requirements programmatically and in the console

This example shows how you might create an identity-based policy that allows a user to view and update their account's password requirements. The password requirements specify the complexity requirements and mandatory rotation periods for the account members' passwords. This policy defines permissions for programmatic and console access.

To learn how to set the account password requirements policy for your account, see Set an account password policy for IAM users.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GetAccountPasswordPolicy",
            "iam:UpdateAccountPasswordPolicy"
        ],
        "Resource": "*"
    }
}
```

IAM: Access the policy simulator API based on user path

This example shows how you might create an identity-based policy that allows using the policy simulator API only for those users that have the path `Department/Development`. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:GetContextKeysForPrincipalPolicy",
                "iam:SimulatePrincipalPolicy"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:user/Department/Development/*"
        }
    ]
}
```

Note

To create a policy that allows using the policy simulator console for those users that have the path `Department/Development`, see IAM: Access the policy simulator console based on user path.

IAM: Access the policy simulator console based on user path

This example shows how you might create an identity-based policy that allows using the policy simulator console only for those users that have the path `Department/Development`. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:GetPolicy",
                "iam:GetUserPolicy"
```

```
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "iam:GetUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListGroupsForUser",
                "iam:ListUserPolicies",
                "iam:ListUsers"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:user/Department/Development/*"
        }
    ]
}
```

IAM: Allows IAM users to self-manage an MFA device

This example shows how you might create an identity-based policy that allows IAM users to self-manage their multi-factor authentication (MFA) device. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI.

> **Note**
>
> If an IAM user with this policy is not MFA-authenticated, this policy denies access to all AWS actions except those necessary to authenticate using MFA. If you add these permissions for a user that is signed in to AWS, they might need to sign out and back in to see these changes.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowListActions",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:ListUsers",
                "iam:ListVirtualMFADevices"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowUserToCreateVirtualMFADevice",
            "Effect": "Allow",
            "Action": [
                "iam:CreateVirtualMFADevice"
            ],
            "Resource": "arn:aws:iam::*:mfa/*"
        },
        {
            "Sid": "AllowUserToManageTheirOwnMFA",
            "Effect": "Allow",
            "Action": [
                "iam:EnableMFADevice",
                "iam:GetMFADevice",
                "iam:ListMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        },
        {
            "Sid": "AllowUserToDeactivateTheirOwnMFAOnlyWhenUsingMFA",
            "Effect": "Allow",
            "Action": [
                "iam:DeactivateMFADevice"
            ],
            "Resource": [
                "arn:aws:iam::*:user/${aws:username}"
            ],
            "Condition": {
                "Bool": {
                    "aws:MultiFactorAuthPresent": "true"
                }
            }
        },
        {
            "Sid": "BlockMostAccessUnlessSignedInWithMFA",
            "Effect": "Deny",
            "NotAction": [
```

```
                "iam:CreateVirtualMFADevice",
                "iam:EnableMFADevice",
                "iam:ListMFADevices",
                "iam:ListUsers",
                "iam:ListVirtualMFADevices",
                "iam:ResyncMFADevice"
            ],
            "Resource": "*",
            "Condition": {
                "BoolIfExists": {
                    "aws:MultiFactorAuthPresent": "false"
                }
            }
        }
    ]
}
```

IAM: Allows IAM users to update their own credentials programmatically and in the console

This example shows how you might create an identity-based policy that allows IAM users to update their own access keys, signing certificates, service specific credentials, and passwords. This policy defines permissions for programmatic and console access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListUsers",
                "iam:GetAccountPasswordPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:*AccessKey*",
```

```
                "iam:ChangePassword",
                "iam:GetUser",
                "iam:*ServiceSpecificCredential*",
                "iam:*SigningCertificate*"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        }
    ]
}
```

To learn how a user can change their own password in the console, see the section called "How an IAM user changes their own password".

IAM: View service last accessed information for an AWS Organizations policy

This example shows how you might create an identity-based policy that allows viewing service last accessed information for a specific AWS Organizations policy. This policy allows retrieving data for the service control policy (SCP) with the `p-policy123` ID. The person who generates and views the report must be authenticated using AWS Organizations management account credentials. This policy allows the requester to retrieve the data for any AWS Organizations entity in their organization. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

For important information about last accessed information, including permissions required, troubleshooting, and supported Regions, see Refine permissions in AWS using last accessed information.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOrgsReadOnlyAndIamGetReport",
            "Effect": "Allow",
            "Action": [
                "iam:GetOrganizationsAccessReport",
                "organizations:Describe*",
                "organizations:List*"
```

```
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowGenerateReportOnlyForThePolicy",
            "Effect": "Allow",
            "Action": "iam:GenerateOrganizationsAccessReport",
            "Resource": "*",
            "Condition": {
                "StringEquals": {"iam:OrganizationsPolicyId": "p-policy123"}
            }
        }
    ]
}
```

IAM: Limits managed policies that can be applied to an IAM user, group, or role

This example shows how you might create an identity-based policy that limits customer managed and AWS managed policies that can be applied to an IAM user, group, or role. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:AttachUserPolicy",
            "iam:DetachUserPolicy"
        ],
        "Resource": "*",
        "Condition": {
            "ArnEquals": {
                "iam:PolicyARN": [
                    "arn:aws:iam::*:policy/policy-name-1",
                    "arn:aws:iam::*:policy/policy-name-2"
                ]
            }
```

```
        }
    }
}
```

AWS: Deny access to resources outside your account except AWS managed IAM policies

Using `aws:ResourceAccount` in your identity-based policies can impact the user or the role's ability to utilize some services that require interaction with resources in accounts owned by a service.

You can create a policy with an exception to allow for AWS managed IAM policies. A service-managed account outside of your AWS Organizations owns Managed IAM Policies. There are four IAM actions that list and retrieve AWS-managed policies. Use these actions in the `NotAction` element of the statement. `AllowAccessToResourcesInSpecificAccountsAndSpecificService1` in the policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToResourcesInSpecificAccountsAndSpecificService1",
      "Effect": "Deny",
      "NotAction": [
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:ListEntitiesForPolicy",
        "iam:ListPolicies"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "111122223333"
          ]
        }
      }
    }
```

```
  ]
}
```

AWS Lambda: Allows a Lambda function to access an Amazon DynamoDB table

This example shows how you might create an identity-based policy that allows read and write access to a specific Amazon DynamoDB table. The policy also allows writing log files to CloudWatch Logs. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

To use this policy, attach the policy to a Lambda service role. A service role is a role that you create in your account to allow a service to perform actions on your behalf. That service role must include AWS Lambda as the principal in the trust policy. For details about how to use this policy, see How to Create an AWS IAM Policy to Grant AWS Lambda Access to an Amazon DynamoDB Table in the AWS Security Blog.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadWriteTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/SampleTable"
        },
        {
            "Sid": "GetStreamRecords",
            "Effect": "Allow",
            "Action": "dynamodb:GetRecords",
            "Resource": "arn:aws:dynamodb:*:*:table/SampleTable/stream/* "
```

```
        },
        {
            "Sid": "WriteLogStreamsAndGroups",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CreateLogGroup",
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "*"
        }
    ]
}
```

Amazon RDS: Allows full RDS database access within a specific Region

This example shows how you might create an identity-based policy that allows full RDS database access within a specific Region. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds:*",
            "Resource": ["arn:aws:rds:us-east-1:*:*"]
        },
        {
            "Effect": "Allow",
            "Action": ["rds:Describe*"],
            "Resource": ["*"]
```

```
        }
    ]
}
```

Amazon RDS: Allows restoring RDS databases, programmatically and in the console

This example shows how you might create an identity-based policy that allows restoring RDS databases. This policy defines permissions for programmatic and console access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSnapshot",
                "rds:DeleteDBSnapshot",
                "rds:Describe*",
                "rds:DownloadDBLogFilePortion",
                "rds:List*",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyOptionGroup",
                "rds:RebootDBInstance",
                "rds:RestoreDBInstanceFromDBSnapshot",
                "rds:RestoreDBInstanceToPointInTime"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon RDS: Allows tag owners full access to RDS resources that they have tagged

This example shows how you might create an identity-based policy that allows tag owners full access to RDS resources that they have tagged. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:Describe*",
                "rds:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "rds:DeleteDBInstance",
                "rds:RebootDBInstance",
                "rds:ModifyDBInstance"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:db-tag/Owner": "${aws:username}"}
            }
        },
        {
            "Action": [
                "rds:ModifyOptionGroup",
                "rds:DeleteOptionGroup"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:og-tag/Owner": "${aws:username}"}
            }
```

```
        },
        {
            "Action": [
                "rds:ModifyDBParameterGroup",
                "rds:ResetDBParameterGroup"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:pg-tag/Owner": "${aws:username}"}
            }
        },
        {
            "Action": [
                "rds:AuthorizeDBSecurityGroupIngress",
                "rds:RevokeDBSecurityGroupIngress",
                "rds:DeleteDBSecurityGroup"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:secgrp-tag/Owner":
"${aws:username}"}
            }
        },
        {
            "Action": [
                "rds:DeleteDBSnapshot",
                "rds:RestoreDBInstanceFromDBSnapshot"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:snapshot-tag/Owner":
"${aws:username}"}
            }
        },
        {
            "Action": [
                "rds:ModifyDBSubnetGroup",
                "rds:DeleteDBSubnetGroup"
            ],
            "Effect": "Allow",
            "Resource": "*",
```

```
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:subgrp-tag/Owner":
 "${aws:username}"}
            }
        },
        {
            "Action": [
                "rds:ModifyEventSubscription",
                "rds:AddSourceIdentifierToSubscription",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:DeleteEventSubscription"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {"rds:es-tag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

Amazon S3: Allows Amazon Cognito users to access objects in their bucket

This example shows how you might create an identity-based policy that allows Amazon Cognito users to access objects in a specific Amazon S3 bucket. This policy allows access only to objects with a name that includes `cognito`, the name of the application, and the federated principal ID, represented by the ${cognito-identity.amazonaws.com:sub} variable. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

> **Note**
>
> The 'sub' value used in the object key is not the user's sub value in the User Pool, it is the identity id associated with the user in the Identity Pool.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ListYourObjects",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": [
        "arn:aws:s3:::bucket-name"
      ],
      "Condition": {
        "StringLike": {
          "s3:prefix": [
            "cognito/application-name/${cognito-identity.amazonaws.com:sub}/*"
          ]
        }
      }
    },
    {
      "Sid": "ReadWriteDeleteYourObjects",
      "Effect": "Allow",
      "Action": [
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-name/cognito/application-name/${cognito-
identity.amazonaws.com:sub}/*"
      ]
    }
  ]
}
```

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. Your users can sign in directly with a user name and password, or through a third party such as Facebook, Amazon, or Google.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.

For more information about Amazon Cognito, see Amazon Cognito User Guide.

Amazon S3: Allows federated users access to their Amazon S3 home directory, programmatically and in the console

This example shows how you might create an identity-based policy that allows federated principals to access their own home directory bucket object in S3. The home directory is a bucket that includes a home folder and folders for individual federated principals. This policy defines permissions for programmatic and console access. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The `${aws:userid}` variable in this policy resolves to `role-id:specified-name`. The `role-id` part of the federated principal ID is a unique identifier assigned to the federated principal's role during creation. For more information, see Unique identifiers. The `specified-name` is the RoleSessionName parameter passed to the `AssumeRoleWithWebIdentity` request when the federated principal assumed their role.

You can view the role ID using the AWS CLI command `aws iam get-role --role-name` *`specified-name`*. For example, imagine that you specify the friendly name `John` and the CLI returns the role ID `AROAXXT2NJT7D3SIQN7Z6`. In this case, the federated principal's user ID is `AROAXXT2NJT7D3SIQN7Z6:John`. This policy then allows the federated principal John to access the Amazon S3 bucket with prefix `AROAXXT2NJT7D3SIQN7Z6:John`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "S3ConsoleAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock",
```

```
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetBucketPolicyStatus",
                "s3:GetBucketPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "",
                        "home/",
                        "home/${aws:userid}/*"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:userid}",
                "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:userid}/*"
            ]
        }
    ]
}
```

Amazon S3: S3 Bucket access, but production bucket denied without recent MFA

This example shows how you might create an identity-based policy that allows an Amazon S3 administrator to access any bucket, including updating, adding, and deleting objects. However, it explicitly denies access to the `amzn-s3-demo-bucket-production` bucket if the user has not signed in using multi-factor authentication (MFA) within the last thirty minutes. This policy grants the permissions necessary to perform this action in the console or programmatically using the AWS

CLI or AWS API. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

This policy never allows programmatic access to the `amzn-s3-demo-bucket` bucket using long-term user access keys. This is accomplished using the `aws:MultiFactorAuthAge` condition key with the `NumericGreaterThanIfExists` condition operator. This policy condition returns `true` if MFA is not present or if the age of the MFA is greater than 30 minutes. In those situations, access is denied. To access the `amzn-s3-demo-bucket-production` bucket programmatically, the S3 administrator must use temporary credentials that were generated in the last 30 minutes using the GetSessionToken API operation.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListAllS3Buckets",
            "Effect": "Allow",
            "Action": ["s3:ListAllMyBuckets"],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "AllowBucketLevelActions",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "AllowBucketObjectActions",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:DeleteObject"
            ],
```

```
            "Resource": "arn:aws:s3:::*/*"
        },
        {
            "Sid": "RequireMFAForProductionBucket",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket-production/*",
                "arn:aws:s3:::amzn-s3-demo-bucket-production"
            ],
            "Condition": {
                "NumericGreaterThanIfExists": {"aws:MultiFactorAuthAge": "1800"}
            }
        }
    ]
}
```

Amazon S3: Allows IAM users access to their S3 home directory, programmatically and in the console

This example shows how you might create an identity-based policy that allows IAM users to access their own home directory bucket object in S3. The home directory is a bucket that includes a home folder and folders for individual users. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

This policy will not work when using IAM roles because the `aws:username` variable is not available when using IAM roles. For details about principal key values, see Principal key values.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "S3ConsoleAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock",
                "s3:GetBucketAcl",
```

```
                "s3:GetBucketLocation",
                "s3:GetBucketPolicyStatus",
                "s3:GetBucketPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "",
                        "home/",
                        "home/${aws:username}/*"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:username}",
                "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:username}/*"
            ]
        }
    ]
}
```

Amazon S3: Restrict management to a specific S3 bucket

This example shows how you might create an identity-based policy that restricts management of an Amazon S3 bucket to that specific bucket. This policy grants permission to perform all Amazon S3 actions, but deny access to every AWS service except Amazon S3. See the following example. According to this policy, you can only access Amazon S3 actions that you can perform on an S3 bucket or S3 object resource. This policy grants the permissions necessary to complete this action

programmatically from the AWS API or AWS CLI. To use this policy, replace the *italicized placeholder text* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

If this policy is used in combination with other policies (such as the AmazonS3FullAccess or AmazonEC2FullAccess AWS managed policies) that allow actions denied by this policy, then access is denied. This is because an explicit deny statement takes precedence over allow statements. For more information, see How AWS enforcement code logic evaluates requests to allow or deny access.

Warning

NotAction and NotResource are advanced policy elements that must be used with care. This policy denies access to every AWS service except Amazon S3. If you attach this policy to a user, any other policies that grant permissions to other services are ignored and access is denied.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        },
        {
            "Effect": "Deny",
            "NotAction": "s3:*",
            "NotResource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
```

```
}
```

Grant read and write access to Amazon S3 bucket objects

This example shows how you might create an identity-based policy that allows `Read` and `Write` access to objects in a specific Amazon S3 bucket. This policy grants the permissions necessary to complete this action programmatically from the AWS API or AWS CLI. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The `s3:*Object` action uses a wildcard as part of the action name. The `AllObjectActions` statement allows the `GetObject`, `DeleteObject`, `PutObject`, and any other Amazon S3 action that ends with the word "Object".

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": ["s3:ListBucket"],
            "Resource": ["arn:aws:s3:::bucket-name"]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object",
            "Resource": ["arn:aws:s3:::bucket-name/*"]
        }
    ]
}
```

Note

To allow `Read` and `Write` access to an object in an Amazon S3 bucket and also include additional permissions for console access, see Amazon S3: Allows read and write access to objects in an S3 Bucket, programmatically and in the console.

Amazon S3: Allows read and write access to objects in an S3 Bucket, programmatically and in the console

This example shows how you might create an identity-based policy that allows `Read` and `Write` access to objects in a specific S3 bucket. This policy defines permissions for programmatic and console access. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

The `s3:*Object` action uses a wildcard as part of the action name. The `AllObjectActions` statement allows the `GetObject`, `DeleteObject`, `PutObject`, and any other Amazon S3 action that ends with the word "Object".

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "S3ConsoleAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetBucketPolicyStatus",
                "s3:GetBucketPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ListObjectsInBucket",
```

```
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket"]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object",
            "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/*"]
        }
    ]
}
```

Manage IAM policies

IAM gives you the tools to create and manage all types of IAM policies (managed policies and inline policies). To add permissions to an IAM identity (IAM user, group, or role), you create a policy, validate the policy, and then attach the policy to the identity. You can attach multiple policies to an identity, and each policy can contain multiple permissions.

Topics

- Additional resources
- Define custom IAM permissions with customer managed policies
- IAM policy validation
- IAM policy testing with the IAM policy simulator
- Adding and removing IAM identity permissions
- Versioning IAM policies
- Edit IAM policies
- Delete IAM policies
- Refine permissions in AWS using last accessed information

Additional resources

The following resources can help you learn more about AWS policies.

- For more information about the different types of IAM policies, see Policies and permissions in AWS Identity and Access Management.
- For general information about using policies within IAM, see Access management for AWS resources.
- For information about how to use IAM Access Analyzer to generate an IAM policy that is based on access activity for an entity, see IAM Access Analyzer policy generation.
- For information about how permissions are evaluated when multiple policies are in effect for a given IAM identity, see Policy evaluation logic.
- The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

Define custom IAM permissions with customer managed policies

Policies define permissions for identities or resources in AWS. You can create *customer managed policies* in IAM using the AWS Management Console, AWS CLI, or AWS API. Customer managed policies are standalone policies that you manage in your own AWS account. You can then attach the policies to identities (users, groups, and roles) in your AWS account.

An *identity-based policy* is a policy attached to an identity in IAM. Identity-based policies can include AWS managed policies, customer managed policies, and inline policies. AWS managed policies are created and managed by AWS, and you can use them but not manage them. An inline policy is one that you create and embed directly to an IAM user group, user, or role. Inline policies can't be reused on other identities or managed outside of the identity where they exist. For more information, see Adding and removing IAM identity permissions.

It's generally better to use customer managed policies instead of inline policies or AWS managed policies. AWS managed policies usually provide broad administrative or read-only permissions. For the greatest security, grant the least privilege, which means granting only the permissions required to perform specific job tasks.

When you create or edit IAM policies, AWS can automatically perform policy validation to help you create an effective policy with least privilege in mind. In the AWS Management Console, IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see IAM policy validation. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation.

You can use the AWS Management Console, AWS CLI, or AWS API to create customer managed policies in IAM. For more information about using CloudFormation templates to add or update policies, see AWS Identity and Access Management resource type reference in the *CloudFormation User Guide*.

Topics

- Create IAM policies (console)
- Create IAM policies (AWS CLI)
- Create IAM policies (AWS API)

Create IAM policies (console)

A policy is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS Management Console to create *customer managed policies* in IAM. Customer managed policies are standalone policies that you administer in your own AWS account. You can then attach the policies to identities (users, groups, and roles) in your AWS account.

The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

Topics

- Creating IAM policies
- Creating policies using the JSON editor
- Creating policies with the visual editor
- Importing existing managed policies

Creating IAM policies

You can create a customer managed policy in the AWS Management Console using one of the following methods:

- **JSON** — Paste and customize a published example identity-based policy.
- **Visual editor** — Construct a new policy from scratch in the visual editor. If you use the visual editor, you do not have to understand JSON syntax.
- **Import** — Import and customize a managed policy from within your account. You can import an AWS managed policy or a customer managed policy that you previously created.

The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

Creating policies using the JSON editor

You can type or paste policies in JSON by choosing the **JSON** option. This method is useful for copying an example policy to use in your account. Or, you can type your own JSON policy document in the JSON editor. You can also use the **JSON** option to toggle between the visual editor and JSON to compare the views.

When you create or edit a policy in the JSON editor, IAM performs policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with actionable recommendations to help you further refine the policy.

A JSON policy document consists of one or more statements. Each statement should contain all the actions that share the same effect (`Allow` or `Deny`) and support the same resources and conditions. If one action requires you to specify all resources (`"*"`) and another action supports the Amazon Resource Name (ARN) of a specific resource, they must be in two separate JSON statements. For details about ARN formats, see Amazon Resource Name (ARN) in the *AWS General Reference Guide*. For general information about IAM policies, see Policies and permissions in AWS Identity and Access Management. For information about the IAM policy language, see IAM JSON policy reference.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.
3. Choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Type or paste a JSON policy document. For details about the IAM policy language, see IAM JSON policy reference.
6. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

7. (Optional) When you create or edit a policy in the AWS Management Console, you can generate a JSON or YAML policy template that you can use in CloudFormation templates.

   To do this, in the **Policy editor** choose **Actions**, and then choose **Generate CloudFormation template**. To learn more about CloudFormation see AWS Identity and Access Management resource type reference in the AWS CloudFormation User Guide.
8. When you are finished adding permissions to the policy, choose **Next**.
9. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
10. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
11. Choose **Create policy** to save your new policy.

After you create a policy, you can attach it to your groups, users, or roles. For more information, see Adding and removing IAM identity permissions.

Creating policies with the visual editor

The visual editor in the IAM console guides you through creating a policy without having to write JSON syntax. To view an example of using the visual editor to create a policy, see the section called "Controlling access to identities".

To use the visual editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.
3. Choose **Create policy**.

4. In the **Policy editor** section, find the **Select a service** section, and then choose an AWS service. You can use the search box at the top to limit the results in the list of services. You can choose only one service within a visual editor permission block. To grant access to more than one service, add multiple permission blocks by choosing **Add more permissions**.
5. In **Actions allowed**, choose the actions to add to the policy. You can choose actions in the following ways:

   - Select the checkbox for all actions.
   - Choose **add actions** to type the name of a specific action. You can use wildcards (*) to specify multiple actions.
   - Select one of the **Access level** groups to choose all actions for the access level (for example, **Read**, **Write**, or **List**).
   - Expand each of the **Access level** groups to choose individual actions.

   By default, the policy that you are creating allows the actions that you choose. To deny the chosen actions instead, choose **Switch to deny permissions**. Because IAM denies by default, we recommend as a security best practice that you allow permissions to only those actions and resources that a user needs. You should create a JSON statement to deny permissions only if you want to override a permission separately allowed by another statement or policy. We recommend that you limit the number of deny permissions to a minimum because they can increase the difficulty of troubleshooting permissions.
6. For **Resources**, if the service and actions that you selected in the previous steps do not support choosing specific resources, all resources are allowed and you cannot edit this section.

   If you chose one or more actions that support resource-level permissions, then the visual editor lists those resources. You can then expand **Resources** to specify resources for your policy.

   You can specify resources in the following ways:

   - Choose **Add ARNs** to specify resources by their Amazon Resource Names (ARN). You can use the visual ARN editor or list ARNs manually. For more information about ARN syntax, see Amazon Resource Name (ARN) in the *AWS General Reference Guide*. For information about using ARNs in the `Resource` element of a policy, see IAM JSON policy elements: Resource.
   - Choose **Any in this account** next to a resource to grant permissions to any resources of that type.

- Choose **All** to choose all resources for the service.

7. (Optional) Choose **Request conditions - *optional*** to add conditions to the policy that you are creating. Conditions limit a JSON policy statement's effect. For example, you can specify that a user is allowed to perform the actions on the resources only when that user's request happens within a certain time range. You can also use commonly used conditions to limit whether a user must be authenticated using a multi-factor authentication (MFA) device. Or you can require that the request originate from within a certain range of IP addresses. For lists of all of the context keys that you can use in a policy condition, see Actions, resources, and condition keys for AWS services in the *Service Authorization Reference*.

   You can choose conditions in the following ways:

   - Use checkboxes to select commonly used conditions.
   - Choose **Add another condition** to specify other conditions. Choose the condition's **Condition Key**, **Qualifier**, and **Operator**, and then type a **Value**. To add more than one value, choose **Add**. You can consider the values as being connected by a logical "OR" operator. When you are finished, choose **Add condition**.

   To add more than one condition, choose **Add another condition** again. Repeat as needed. Each condition applies only to this one visual editor permission block. All the conditions must be true for the permission block to be considered a match. In other words, consider the conditions to be connected by a logical "AND" operator.

   For more information about the **Condition** element, see IAM JSON policy elements: Condition in the IAM JSON policy reference.

8. To add more permission blocks, choose **Add more permissions**. For each block, repeat steps 2 through 5.

   > **Note**
   >
   > You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

9. (Optional) When you create or edit a policy in the AWS Management Console, you can generate a JSON or YAML policy template that you can use in CloudFormation templates.

To do this, in the **Policy editor** choose **Actions**, and then choose **Generate CloudFormation template**. To learn more about CloudFormation see AWS Identity and Access Management resource type reference in the AWS CloudFormation User Guide.

10. When you are finished adding permissions to the policy, choose **Next**.
11. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. Review the **Permissions defined in this policy** to make sure that you have granted the intended permissions.
12. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources.
13. Choose **Create policy** to save your new policy.

After you create a policy, you can attach it to your groups, users, or roles. For more information, see Adding and removing IAM identity permissions.

Importing existing managed policies

An easy way to create a new policy is to import an existing managed policy within your account that has at least some of the permissions that you need. You can then customize the policy to match it to your new requirements.

You cannot import an inline policy. To learn about the difference between managed and inline policies, see Managed policies and inline policies.

To import an existing managed policy in the visual editor

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.
3. Choose **Create policy**.
4. In the **Policy editor** , choose **Visual** and then on the right side of the page, choose **Actions** then choose **Import policy**.
5. In the **Import policy** window, choose the managed policies that most closely match the policy that you want to include in your new policy. You can use the search box at the top to limit the results in the list of policies.

6. Choose **Import policy**.

   The imported policies are added in new permission blocks at the bottom of your policy.

7. Use the **Visual** editor or choose **JSON** to customize your policy. Then choose **Next**.

   > **Note**
   >
   > You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

8. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. You cannot edit these settings later. Review the **Permissions defined in this policy** and then choose **Create policy** to save your work.

To import an existing managed policy in the JSON editor

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.
3. Choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option, and then on the right side of the page, choose **Actions** then choose **Import policy**.
5. In the **Import policy** window, choose the managed policies that most closely match the policy that you want to include in your new policy. You can use the search box at the top to limit the results in the list of policies.
6. Choose **Import policy**.

   Statements from the imported policies are added to the bottom of your JSON policy.

7. Customize your policy in JSON. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**. Or, customize your policy in the **Visual** editor. Then choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

8. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. You cannot edit these later. Review the policy **Permissions defined in this policy** and then choose **Create policy** to save your work.

After you create a policy, you can attach it to your groups, users, or roles. For more information, see Adding and removing IAM identity permissions.

Create IAM policies (AWS CLI)

A policy is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS CLI to create *customer managed policies* in IAM. Customer managed policies are standalone policies that you administer in your own AWS account. As a best practice, we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. By validating your policies you can address any errors or recommendations before you attach the policies to identities (users, groups, and roles) in your AWS account.

The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

Creating IAM policies (AWS CLI)

You can create an IAM customer managed policy or an inline policy using the AWS Command Line Interface (AWS CLI).

To create a customer managed policy (AWS CLI)

Use the following command:

- create-policy

To create an inline policy for an IAM identity (group, user or role) (AWS CLI)

Use one of the following commands:

- put-group-policy
- put-role-policy
- put-user-policy

> **Note**
>
> You can't use IAM to embed an inline policy for a *service-linked role*.

To validate a customer managed policy (AWS CLI)

Use the following IAM Access Analyzer command:

- validate-policy

Create IAM policies (AWS API)

A policy is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS API to create *customer managed policies* in IAM. Customer managed policies are standalone policies that you administer in your own AWS account. As a best practice, we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. By validating your policies you can address any errors or recommendations before you attach the policies to identities (users, groups, and roles) in your AWS account.

The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

Creating IAM policies (AWS API)

You can create an IAM customer managed policy or an inline policy using the AWS API.

To create a customer managed policy (AWS API)

Call the following operation:

- CreatePolicy

To create an inline policy for an IAM identity (group, user, or role) (AWS API)

Call one of the following operations:

- PutGroupPolicy
- PutRolePolicy
- PutUserPolicy

> **Note**
>
> You can't use IAM to embed an inline policy for a *service-linked role*.

To validate a customer managed policy (AWS API)

Call the following IAM Access Analyzer operation:

- ValidatePolicy

IAM policy validation

A policy is a JSON document that uses the IAM policy grammar. When you attach a policy to an IAM entity, such as a user, group, or role, it grants permissions to that entity.

When you create or edit IAM access control policies using the AWS Management Console, AWS automatically examines them to ensure that they comply with the IAM policy grammar. If AWS determines that a policy is not in compliance with the grammar, it prompts you to fix the policy.

IAM Access Analyzer provides additional policy checks with recommendations to help you further refine the policy. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation. To view a list of warnings, errors, and suggestions that are returned by IAM Access Analyzer, see IAM Access Analyzer policy check reference.

Validation scope

AWS checks JSON policy syntax and grammar. It also verifies that your ARNs are formatted properly and action names and condition keys are correct.

Accessing policy validation

Policies are validated automatically when you create a JSON policy or edit an existing policy in the AWS Management Console. If the policy syntax is not valid, you receive a notification and must fix the problem before you can continue. The findings from the IAM Access Analyzer policy validation are automatically returned in the AWS Management Console if you have permissions for `access-analyzer:ValidatePolicy`. You can also validate policies using the AWS API or AWS CLI.

Existing policies

You might have existing policies that are not valid because they were created or last saved before the latest updates to the policy engine. As a best practice, we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. We recommend that you open your existing policies and review the policy validation results that are generated. You cannot edit and save existing policies without fixing any policy syntax errors.

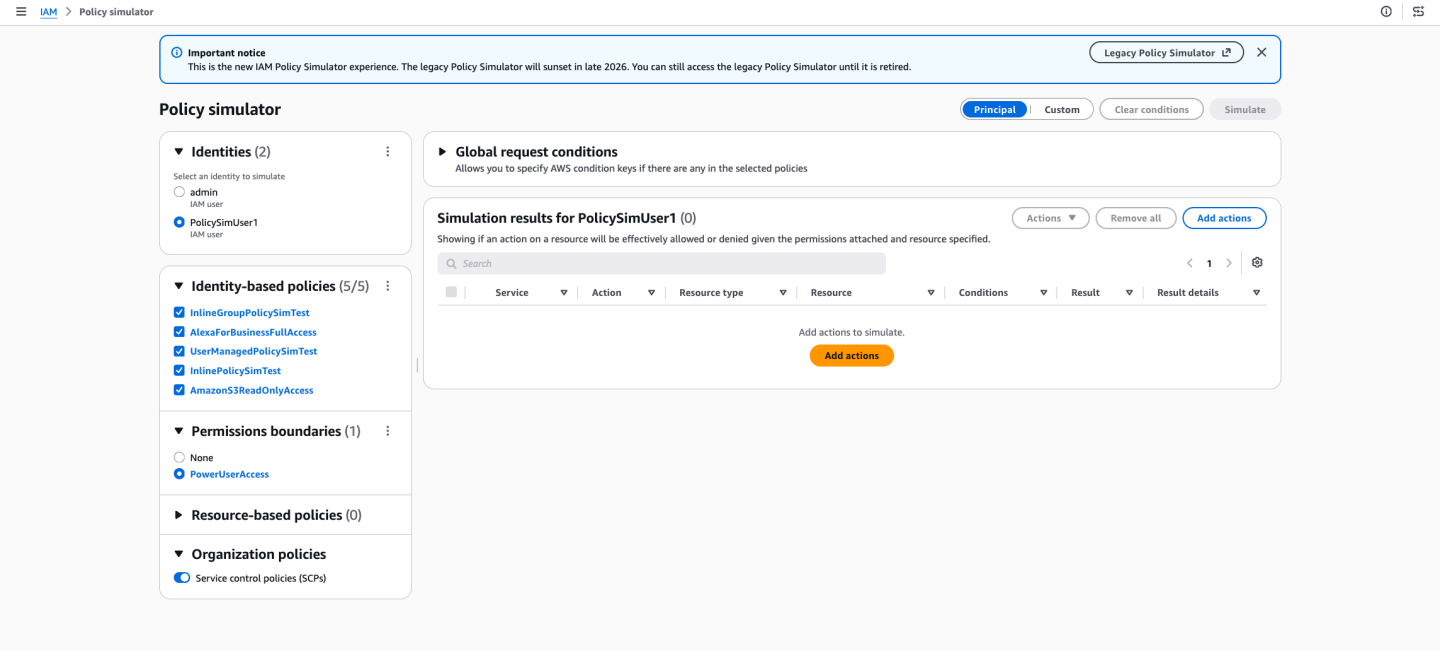
IAM policy testing with the IAM policy simulator

With the IAM policy simulator, you can test identity-based policies, IAM permissions boundaries, service control policies (SCPs), and resource-based policies that you provide. You simulate how AWS would evaluate a request against those policies, without sending a real request to any AWS service. Each simulation returns a binary authorization outcome for every action and resource you test: the action is either allowed or denied. When the result is an allow or explicit deny, the simulator also shows you which policy produced that outcome, so you can understand and fix the cause.

For more information about how and why to use IAM policies, see Policies and permissions in AWS Identity and Access Management.

> **Important**
>
> The policy simulator results can differ from your live AWS environment. We recommend that you check your policies against your live AWS environment after testing with the policy simulator to confirm that you have the desired results. For more information, see How the IAM policy simulator works.

The following screenshot shows the IAM policy simulator, where you select an identity and its policies and choose the actions and resources to test before running the simulation.



Topics

- Policy simulator modes
- What you can do with the policy simulator
- How the IAM policy simulator works
- How condition key evaluation works in the IAM policy simulator
- Permissions for using the policy simulator
- How to simulate policies

Policy simulator modes

The policy simulator has two modes. Choose the mode that matches whether you want to test policies that already exist in your account, or policies that you are still drafting.

Principal mode

In Principal mode, you test the policies that are already attached to an existing IAM user, role, or group. You choose one identity for the simulation, and the simulator evaluates the policies attached to that identity. You can also include or exclude custom identity policies or permissions boundaries to the simulation. Those additions are simulated as if attached to the principal, without actually being attached in IAM, so your account is unchanged.

Custom mode

In Custom mode, you test one or more policies that you write or paste in. This is useful for checking a policy before you attach it. Custom policies are simulated on their own and are not saved to your account.

For detailed, step-by-step instructions on selecting a mode and running a simulation, see How to simulate policies.

What you can do with the policy simulator

Here are some common things you can do with the policy simulator. More detailed step-by-step guide and distinction between API and console behaviors can be found in How to simulate policies.

- Test identity-based policies, whether they are already attached to an IAM user, group, or role, or new policies that you write or paste in. For an attached identity with more than one policy, you can test all the policies together or select individual policies, and see which actions are allowed or denied for specific resources. New policies are evaluated as if they were attached to an identity, without actually attaching them in your account; they are used only in the simulation and are not saved to your account.
- Test the effect of restrictive controls on access. You can simulate one permissions boundary at a time. If your AWS account is a member of an organization in AWS Organizations, you can also test the impact of service control policies (SCPs) on your identity-based policies. The simulator tells you whether an SCP allows or denies an action, but for security reasons it does not surface the matched statements from an SCP the way it does for other policy types.
- Test the effect of a resource-based policy that you provide. Except in the console, the simulator does not retrieve a resource's policy for you. To include a resource-based policy in a simulation, you supply the policy along with the resource.
- Test specific services, actions, and resources under real-world conditions, and identify what determines the outcome. For example, you can test whether a policy allows the `ListAllMyBuckets`, `CreateBucket`, and `DeleteBucket` actions in Amazon S3 on a specific bucket, and provide context keys such as an IP address or date that are referenced in `Condition` elements in the policies being tested. For each result, the simulator identifies which specific statement allows or denies access; matched statements can come from identity-based policies and from resource-based policies that you provide.

How the IAM policy simulator works

When you run a simulation, the policy simulator evaluates the policies in scope for the request, any SCPs in scope, and the inputs that you provide, such as actions, resources, and context key values. In Custom mode you supply the policies directly. In Principal mode the simulator uses the policies attached to the identity you select, and you can include or exclude more policies to the simulation. The result is a binary outcome for each action and resource: allowed or denied.

The policy simulator differs from the live AWS environment in the following ways:

- The policy simulator does not make an actual AWS service request, so you can safely test requests that might otherwise make unwanted changes to your live AWS environment. It does not use real context key values from production, and because the actions are not actually run, it returns no service response. The only result is whether each requested action would be allowed or denied.
- If you edit a policy in the policy simulator, the change affects only the simulation. The corresponding policy in your AWS account remains unchanged.
- The policy simulator evaluates SCPs, including condition keys and resource scoping in deny statements, but does not support resource control policies (RCPs). Simulation results can still differ from live behavior for some advanced configurations, such as VPC endpoint policies, role chaining, or multiple resource-based policies on a single resource.
- Cross-account simulations return per-policy-type decisions (identity policy and resource policy) in `EvalDecisionDetails`.

How condition key evaluation works in the IAM policy simulator

Many policies use `Condition` elements that depend on condition keys, such as `aws:RequestedRegion` or `aws:MultiFactorAuthPresent`. When you run a simulation, the policy simulator gets values for the condition keys from two sources: values that the simulator populates automatically, and values that you provide as simulation input. If a value does not satisfy a condition, the action is denied.

Context that the simulator populates automatically

The simulator populates the following principal and organization context keys for the evaluation:

- Principal context: `aws:PrincipalAccount`, `aws:PrincipalId`, `aws:PrincipalType`, `aws:UserId`, `aws:UserName`

- Organization context: `aws:PrincipalOrgID`, `aws:PrincipalOrgMasterAccountId`, `aws:PrincipalOrgPaths`

Because these keys are populated for you, SCPs and other policies that use principal or organization conditions are evaluated correctly. You provide the values for all other condition keys that your policies reference. Review the `Condition` elements in the policies you are testing and supply values for any keys that are not in the list above.

Global condition keys and service support

For identity-based policies and resource-based policies, the simulator does not determine which services support a given global condition key for authorization. For example, the simulator does not identify that a service does not support `aws:TagKeys`. In contrast, service control policies (SCPs) are evaluated with service-aware logic, including global condition keys such as `aws:RequestedRegion` and `aws:PrincipalAccount`. For more information about condition keys, see AWS global condition context keys.

Condition keys in service control policies (SCPs)

The simulator evaluates the condition keys and resource scoping in SCPs. How you supply values for those keys depends on how you run the simulation:

- With the AWS CLI or AWS API, you can provide values for condition keys that are referenced by SCPs.
- In the IAM console, you can only set values for condition keys that appear in your identity-based policies, permissions boundaries, or resource-based policies. If a condition key is referenced only by an SCP, you cannot set a value for it in the console. However, if the same condition key also appears in one of those other policies, the value you set is used across the entire evaluation, including the SCP.

For security reasons, SCP evaluation does not return missing context values. You still receive the allow or deny decision for the request.

Permissions for using the policy simulator

The permissions you need depend on whether you use the AWS CLI or AWS API, or the IAM console, and which mode you use.

Console

You need permission to call `iam:GetContextKeysForCustomPolicy` to use the IAM policy simulator on the console.

Additional permissions you need depend on the mode. In Principal mode you need additional read permissions to list the identities in your account and read their attached policies. To reduce what you have to grant, use Custom mode if you only need to test policies that you paste in.

Custom mode

To use Custom mode, where you paste in the policies you want to test, you also need `iam:SimulateCustomPolicy`

Principal mode

To use Principal mode, where the console loads and displays the identities and policies in your account, you need the common permission plus the following. This is the larger set, because the console has to discover identities and read their attached policies.

Simulate:

- `iam:SimulatePrincipalPolicy`

Listing identities:

- `iam:ListUsers`, `iam:ListRoles`, `iam:ListGroups`
- `iam:ListGroupsForUser`

Listing attached policies:

- `iam:ListPolicies`, `iam:ListUserPolicies`, `iam:ListRolePolicies`, `iam:ListGroupPolicies`
- `iam:ListAttachedUserPolicies`, `iam:ListAttachedRolePolicies`, `iam:ListAttachedGroupPolicies`

Getting policy documents:

- `iam:GetPolicyVersion`, `iam:GetUserPolicy`, `iam:GetRolePolicy`, `iam:GetGroupPolicy`

Getting permissions boundaries:

- `iam:GetUser`, `iam:GetRole`

Additional permissions for Principal mode to simulate resource-based policies

To include a resource's policy in a Principal mode simulation, add the read permission for that resource type:

- `s3:GetBucketPolicy`, `sns:GetTopicAttributes`, `sqs:GetQueueUrl`, `sqs:GetQueueAttributes`, `glacier:GetVaultAccessPolicy`

For example policies that grant access to the simulator, see the section called “Example policies: AWS Identity and Access Management (IAM)”.

AWS CLI and AWS API

For the AWS CLI and AWS API, you need only the permission to call the simulation action for the type of policy you want to test. If you use `iam:GetContextKeysForCustomPolicy` and `iam:GetContextKeysForPrincipalPolicy` to discover the context keys first, also grant permissions to these actions. For security reasons, the actions are split into the following two modes. Grant only the permissions that match what you want to simulate.

To simulate policies that you pass directly to the API as strings (custom policies)

- `iam:SimulateCustomPolicy`
- `iam:GetContextKeysForCustomPolicy` (only if you use the context-key discovery step)

These actions let you test policies that are not yet attached to a user, user group, or role. You pass the policies as strings, and they are used only in the simulation.

To simulate policies that are attached to an IAM user, user group, role, or resource

- `iam:SimulatePrincipalPolicy`
- `iam:GetContextKeysForPrincipalPolicy` (only if you use the context-key discovery step)

Because these actions can reveal the permissions assigned to other IAM entities, consider restricting who can call them. For example, to give a user named Bob permission to simulate a policy assigned to a user named Alice, give Bob access to the resource `arn:aws:iam::777788889999:user/alice`.

To view an example policy that allows using the policy simulator API for attached and unattached policies in the current AWS account, see IAM: Access the policy simulator API. To view an example policy scoped to users with a specific path, see IAM: Access the policy simulator API based on user path.

How to simulate policies

You can run simulations in the IAM console or with the AWS CLI and AWS API. For an overview of what the simulator evaluates, the two modes, and how it reaches a result, see IAM policy testing with the IAM policy simulator. For the permissions you need, see Permissions for using the policy simulator.

Topics

- Simulate policies (console)
- Simulate policies (AWS CLI and AWS API)

Simulate policies (console)

You can run simulations from the policy simulator in the IAM console. In the navigation pane, choose **Policy simulator**.

> **Note**
>
> The standalone policy simulator console at https://policysim.aws.amazon.com/ is no longer maintained. Use the policy simulator in the IAM console.

Principal mode

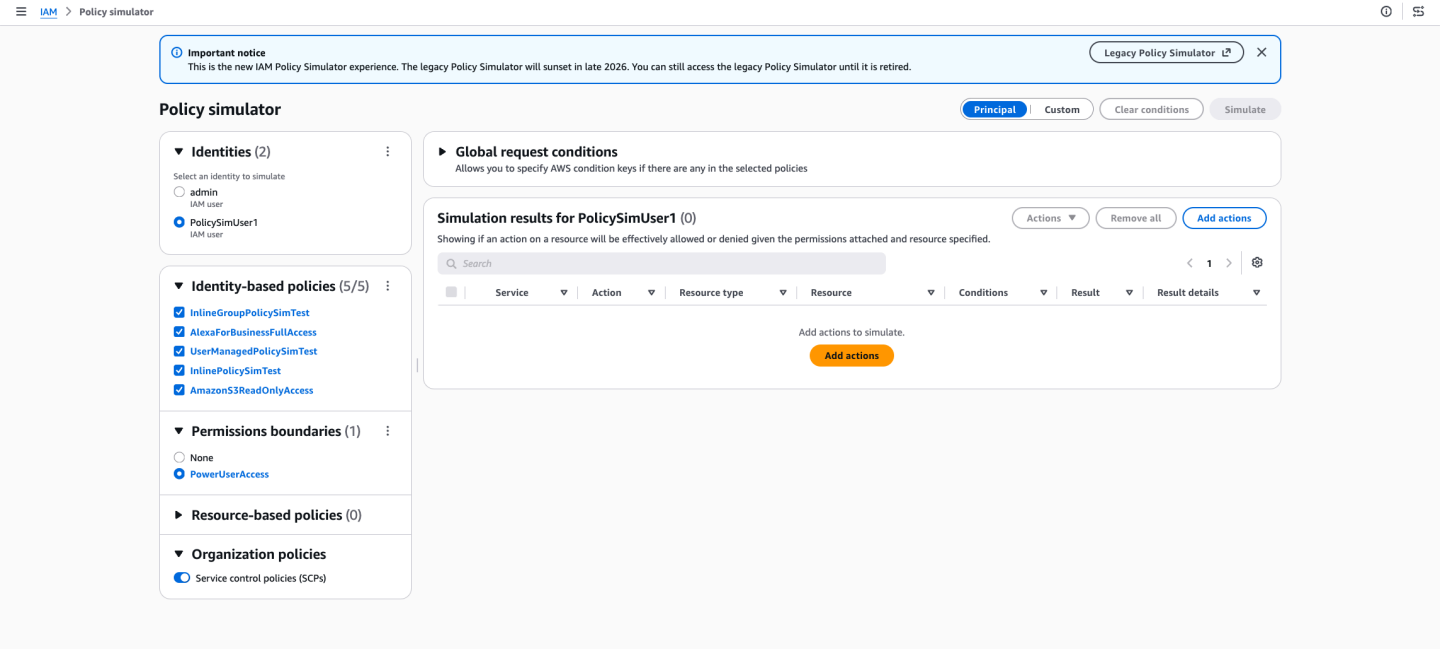
Select an identity

You first start by selecting an identity. You can select IAM users, roles, or groups in that account. The side panel will populate the policies attached to these identities after your selection.

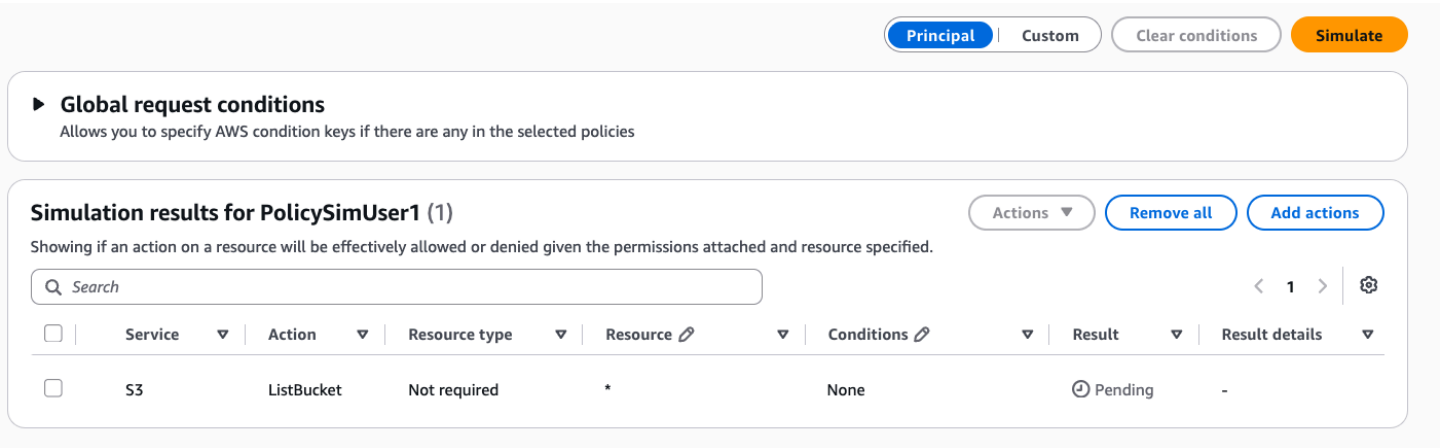
Include or exclude policies

The side panel lists the policies in scope for the simulation, including inline, AWS managed, and customer managed policies, and any permissions boundary. You can exclude a policy to see how the result changes without it, then include it again. Excluding a policy that granted access changes the result to an implicit deny.

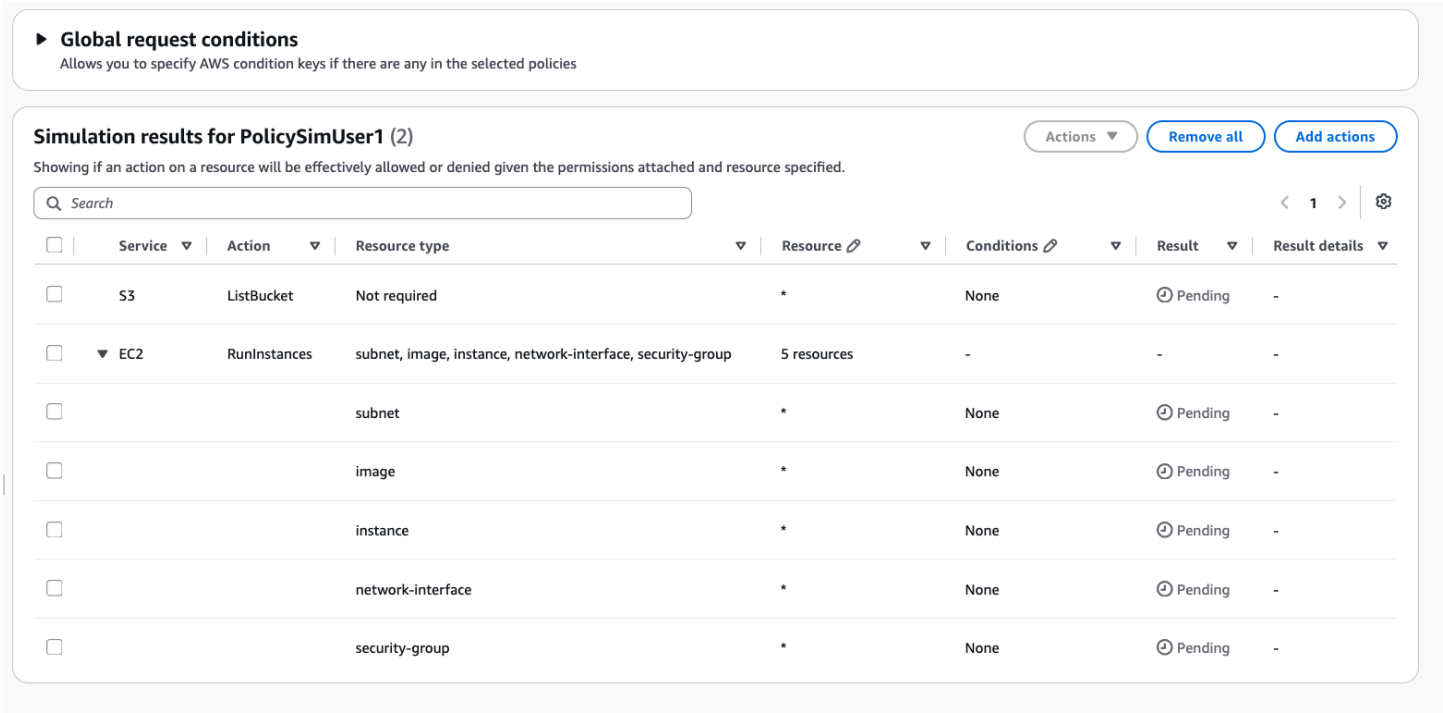
Add actions and resources



After you choose identities and policies in the left panel, add the actions that you want to test. For an action that supports a resource selection, you can test against all resources or enter one or more specific resource ARNs. When several actions use the same resource type, you can apply one resource to all of them at once, or scope it to a single action.



Some actions require a group of related resource types before the simulator can evaluate them. For example, when you simulate `ec2:RunInstances`, the console prompts you to specify an image, an instance, and a security group together, and a subnet or a volume depending on the scenario.



Provide condition key values

If the policies that you test include condition keys, the simulator lists the relevant keys and lets you set values before you run the simulation. The keys shown reflect the policies that are currently selected, so selecting or clearing a policy adds or removes its keys. If a value does not satisfy a condition, the action is denied.

The simulator provides values for condition keys in two places. **Global request conditions** apply to the whole simulated request. These are the keys that begin with `aws:`, such as `aws:MultiFactorAuthPresent` or `aws:PrincipalServiceNamesList`, and you set their values once in the global conditions area. Selecting or clearing a policy adds or removes its global keys, and the values you enter are retained as you adjust which policies are in scope. **Service-specific conditions** apply to a single action. Each action row shows the number of conditions it references, and you open the row to set the service-specific keys for that action, such as `cloudwatch:namespace` for `cloudwatch:PutMetricData` or `ram:PermissionResourceType` for `ram:ListPermissionAssociations`. An action row can include both a global key and a service-specific key, and if any value does not satisfy a condition, that action is denied.

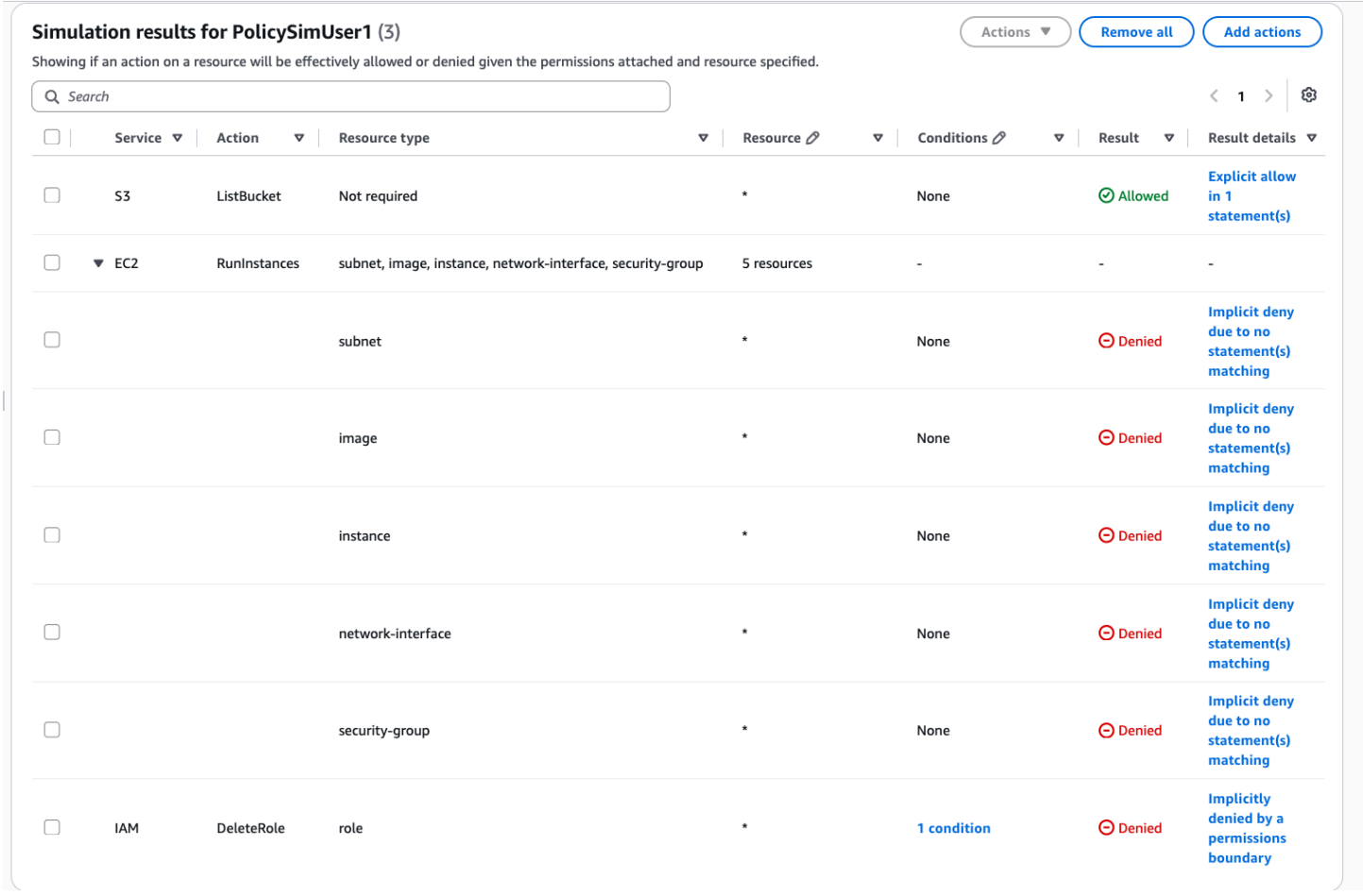
IAM policy simulator populates some principal and organization context keys for you, and you provide values for the rest. For the list of keys that are populated automatically and more about how condition keys are evaluated, see How condition key evaluation works in the IAM policy simulator.

To learn how to use condition keys, see AWS global condition context keys.

Test a permissions boundary

When an identity has a permissions boundary, it appears in the side panel and is included by default. You can also exclude the boundary and run the simulation again, if you want to find out whether the boundary is the reason an action is denied.

For example, suppose you simulate `iam:DeleteRole` for an identity whose boundary does not allow it. The result is a deny attributed to the permissions boundary. If you then add a policy that allows the action and exclude the boundary, the result changes to an allow. This confirms that the boundary was the only control blocking the action.



Include service control policies

If your account is a member of an organization, you can include service control policies (SCPs) in the simulation from the side panel. The simulator evaluates the auto-populated condition keys and resource scoping in SCPs when it determines the result. When an SCP explicitly denies an action, the result is an explicit deny attributed to AWS Organizations, and an allow in another policy cannot override it. Turn the control off and run the simulation again to see the result without the organization's controls. For a list of the auto-populated condition keys, see How condition key evaluation works in the IAM policy simulator.

> **Note**
>
> For security reasons, the simulator does not surface the details of SCP evaluation. Condition keys that are referenced only by SCPs are not listed for you to set values, and the matched statements returned for a result do not include statements from SCPs. The keys and matched statements come from the identity-based policies attached to the identity, and any policies that you provide.

Test a resource-based policy

To include a resource-based policy in a console simulation, enter a specific resource ARN and choose the option to include the resource's policy. The resource-based policy appears in the side panel and is evaluated in the result. In the console, this is available for Amazon S3 buckets, Amazon SQS queues, Amazon SNS topics, and Amazon Glacier vaults.

> **Note**
>
> Simulation of resource-based policies isn't supported for IAM roles.

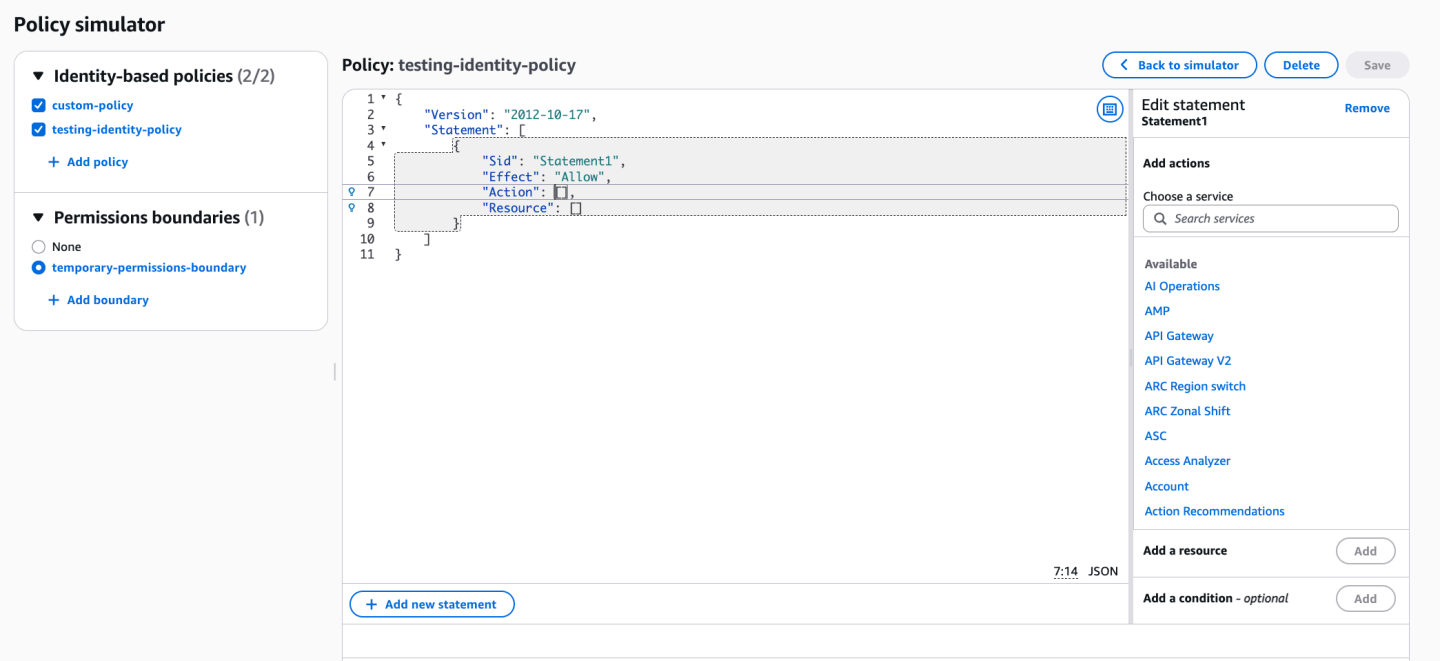
Read results

Each row in the results table shows whether the action is allowed, implicitly denied, or explicitly denied, along with a short explanation of the outcome. Use the explanation to identify which policy or control determined the result, such as an explicit allow, the absence of a matching statement, a permissions boundary, or an organization control.

Custom mode

You can use the IAM policy editor to write or paste in identity-based policies and permission boundaries in custom mode. These policies are not attached to an identity and can be evaluated before you apply them. You can include multiple identity-based policies, but only one permissions boundary for simulation.

> **Note**
>
> In Custom mode, the console does not support passing in custom SCPs or simulating resource-based policies. To simulate custom SCPs or resource-based policies, use the AWS CLI or AWS API.



Simulate policies (AWS CLI and AWS API)

Simulating policies with the AWS CLI or AWS API generally takes two steps:

1. Evaluate the policies and return the list of context keys that they reference, so that you know which context key values to supply. This step is optional.
2. Simulate the policies, providing the actions, resources, and context key values to use during the simulation.

Each action is paired with each resource, and the simulation determines whether the policies allow or deny that action for that resource. You can provide values for any context keys that your policies reference. If you do not provide a value for a context key, the simulation still runs and returns an implicit deny, and the missing keys are returned in `missingContextValues` so you know which values to add. If a missing context key is referenced only by an SCP, that key is not returned for security reasons, but you still receive the decision.

The API actions are split into two groups:

- Actions that simulate only policies passed directly to the API as strings: GetContextKeysForCustomPolicy and SimulateCustomPolicy.
- Actions that simulate the policies attached to a specified IAM user, user group, role, or resource: GetContextKeysForPrincipalPolicy and SimulatePrincipalPolicy. Because these can reveal permissions assigned to other IAM entities, consider restricting access to them.

These actions also support the following:

- `OrderedOrganizationPolicyInputList` on SimulateCustomPolicy lets you simulate the effect of an organization's SCP hierarchy without using a live principal.
- `PolicyExclusionList` on SimulatePrincipalPolicy lets you test "what if I remove this policy?" scenarios without changing which policies are attached to the principal.
- The response returns a single `EvaluationResult` per action, with aggregate decisions at the top level and per-resource details in `ResourceSpecificResults`.

AWS CLI and AWS API reference

For sample AWS CLI commands to simulate policies, see the following:

- AWS CLI: get-context-keys-for-custom-policy, get-context-keys-for-principal-policy, simulate-custom-policy, simulate-principal-policy
- AWS API: GetContextKeysForCustomPolicy, GetContextKeysForPrincipalPolicy, SimulateCustomPolicy, SimulatePrincipalPolicy

Adding and removing IAM identity permissions

You use policies to define the permissions for an identity (user, user group, or role). You can add and remove permissions by attaching and detaching IAM policies for an identity using the AWS

Management Console, the AWS Command Line Interface (AWS CLI), or the AWS API. You can also use policies to set permissions boundaries for only entities (users or roles) that are using the same methods. Permissions boundaries are an advanced AWS feature that control the maximum permissions that an entity can have.

Topics

- Terminology
- View identity activity
- Adding IAM identity permissions (console)
- Removing IAM identity permissions (console)
- Adding IAM policies (AWS CLI)
- Removing IAM policies (AWS CLI)
- Adding IAM policies (AWS API)
- Removing IAM policies (AWS API)

Terminology

When you associate permissions policies with identities (IAM users, IAM groups, and IAM roles), terminology and procedures vary depending on whether you are working with a managed or inline policy:

- **Attach** – Used with managed policies. You attach a managed policy to an identity (a user, user group, or role). Attaching a policy applies the permissions in the policy to the identity.
- **Detach** – Used with managed policies. You detach a managed policy from an IAM identity (a user, user group, or role). Detaching a policy removes its permissions from the identity.
- **Embed** – Used with inline policies. You embed an inline policy in an identity (a user, user group, or role). Embedding a policy applies the permissions in the policy to the identity. Because an inline policy is stored in the identity, it is embedded rather than attached, though the results are similar.

> **Note**
>
> You can embed an inline policy for a *service-linked role* only in the service that depends on the role. See the AWS documentation for your service to see whether it supports this feature.

- **Delete** – Used with inline policies. You delete an inline policy from an IAM identity (a user, user group, or role). Deleting a policy removes its permissions from the identity.

> **Note**
>
> You can delete an inline policy for a *service-linked role* only in the service that depends on the role. See the AWS documentation for your service to see whether it supports this feature.

You can use the console, AWS CLI, or AWS API to perform any of these actions.

More information

- For more information about the difference between managed and inline policies, see Managed policies and inline policies.
- For more information about permissions boundaries, see Permissions boundaries for IAM entities.
- For general information about IAM policies, see Policies and permissions in AWS Identity and Access Management.
- For information about validating IAM policies, see IAM policy validation.
- The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

View identity activity

Before you change the permissions for an identity (user, user group, or role), you should review their recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Adding IAM identity permissions (console)

You can use the AWS Management Console to add permissions to an identity (user, user group, or role). To do this, attach managed policies that control permissions, or specify a policy that serves as a permissions boundary. You can also embed an inline policy.

To use a managed policy as a permissions policy for an identity (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, select the radio button next to the name of the policy to attach. You can use the search box to filter the list of policies.
4. Choose **Actions**, and then choose **Attach**.
5. Select one or more identities to attach the policy to. You can use the search box to filter the list of principal entities. After selecting the identities, choose **Attach policy**.

To use a managed policy to set a permissions boundary (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the name of the policy to set. You can use the search box to filter the list of policies.
4. On the policy details page, choose the **Entities attached** tab, and then, if necessary, open the **Attached as a permissions boundaries** section and choose **Set this policy as a permissions boundary**.
5. Select one or more users or roles on which to use the policy for a permissions boundary. You can use the search box to filter the list of principal entities. After selecting the principals, choose **Set permissions boundary**.

To embed an inline policy for a user or role (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** or **Roles**.
3. In the list, choose the name of the user or role to embed a policy in.
4. Choose the **Permissions** tab.
5. Choose **Add permissions** and then choose **Create inline policy**.

> **Note**
>
> You cannot embed an inline policy in a *service-linked role* in IAM. Because the linked service defines whether you can modify the permissions of the role, you might be able to add additional policies from the service console, API, or AWS CLI. To view the service-linked role documentation for a service, see AWS services that work with IAM and choose **Yes** in the **Service-Linked Role** column for your service.

6. Choose from the following methods to view the steps required to create your policy:

   - Importing existing managed policies – You can import a managed policy within your account and then edit the policy to customize it to your specific requirements. A managed policy can be an AWS managed policy or a customer managed policy that you created previously.
   - Creating policies with the visual editor – You can construct a new policy from scratch in the visual editor. If you use the visual editor, you do not have to understand JSON syntax.
   - Creating policies using the JSON editor – In the **JSON** editor option, you can use JSON syntax to create a policy. You can type a new JSON policy document or paste an example policy.

7. After you create an inline policy, it is automatically embedded in your user or role.

To embed an inline policy for a user group (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **User groups**.
3. In the list, choose the name of the user group to embed a policy in.
4. Choose the **Permissions** tab, choose **Add permissions**, and then choose **Create inline policy**.
5. Do one of the following:

   - Choose the **Visual** option to create the policy. For more information, see Creating policies with the visual editor.
   - Choose the **JSON** option to create the policy. For more information, see Creating policies using the JSON editor.

6. When you are satisfied with the policy, choose **Create policy**.

To change the permissions boundary for one or more entities (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the name of the policy to set. You can use the search box to filter the list of policies.
4. On the policy details page, choose the **Entities attached** tab, and then, if necessary, open the **Attached as a permissions boundary** section. Select the checkbox next to the users or roles whose boundaries you want to change and then choose **Change**.
5. Select a new policy to use for a permissions boundary. You can use the search box to filter the list of policies. After selecting the policy, choose **Set permissions boundary**.

Removing IAM identity permissions (console)

You can use the AWS Management Console to remove permissions from an identity (user, user group, or role). To do this, detach managed policies that control permissions, or remove a policy that serves as a permissions boundary. You can also delete an inline policy.

To detach a managed policy used as a permissions policy (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, select the radio button next to the name of the policy to detach. You can use the search box to filter the list of policies.
4. Choose **Actions**, and then choose **Detach**.
5. Select the identities to detach the policy from. You can use the search box to filter the list of identities. After selecting the identities, choose **Detach policy**.

To remove a permissions boundary (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the name of the policy to set. You can use the search box to filter the list of policies.
4. On the policy summary page, choose the **Entities attached** tab, and then, if necessary, open the **Attached as a permissions boundary** section and choose the entities to remove the permissions boundary from. Then choose **Remove boundary**.
5. Confirm that you want to remove the boundary and choose **Remove boundary**.

To delete an inline policy (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **User groups**, **Users**, or **Roles**.
3. In the list, choose the name of the user group, user, or role that has the policy you want to remove.
4. Choose the **Permissions** tab.
5. Select the checkbox next to the policy and choose **Remove**.
6. Choose **Remove** in the confirmation box.

Adding IAM policies (AWS CLI)

You can use the AWS CLI to add permissions to an identity (user, user group, or role). To do this, attach managed policies that control permissions, or specify a policy that serves as a permissions boundary. You can also embed an inline policy.

To use a managed policy as a permissions policy for an entity (AWS CLI)

1. (Optional) To view information about a managed policy, run the following commands:
   - To list managed policies: aws iam list-policies
   - To retrieve detailed information about a managed policy: get-policy
2. To attach a managed policy to an identity (user, user group, or role), use one of the following commands:
   - aws iam attach-user-policy

- aws iam attach-group-policy
- aws iam attach-role-policy

To use a managed policy to set a permissions boundary (AWS CLI)

1. (Optional) To view information about a managed policy, run the following commands:
   - To list managed policies: aws iam list-policies
   - To retrieve detailed information about a managed policy: aws iam get-policy
2. To use a managed policy to set the permissions boundary for an entity (user or role), use one of the following commands:
   - aws iam put-user-permissions-boundary
   - aws iam put-role-permissions-boundary

To embed an inline policy (AWS CLI)

To embed an inline policy to an identity (user, user group, or role that is not a *service-linked role*), use one of the following commands:

- aws iam put-user-policy
- aws iam put-group-policy
- aws iam put-role-policy

Removing IAM policies (AWS CLI)

You can use the AWS CLI to detach managed policies that control permissions, or remove a policy that serves as a permissions boundary. You can also delete an inline policy.

To detach a managed policy used as a permissions policy (AWS CLI)

1. (Optional) To view information about a policy, run the following commands:
   - To list managed policies: aws iam list-policies
   - To retrieve detailed information about a managed policy: aws iam get-policy
2. (Optional) To find out about the relationships between the policies and identities, run the following commands:

- To list the identities (IAM users, IAM groups, and IAM roles) to which a managed policy is attached:
  - aws iam list-entities-for-policy
- To list the managed policies attached to an identity (a user, user group, or role), use one of the following commands:
  - aws iam list-attached-user-policies
  - aws iam list-attached-group-policies
  - aws iam list-attached-role-policies

3. To detach a managed policy from an identity (user, user group, or role), use one of the following commands:
   - aws iam detach-user-policy
   - aws iam detach-group-policy
   - aws iam detach-role-policy

To remove a permissions boundary (AWS CLI)

1. (Optional) To view which managed policy is currently used to set the permissions boundary for a user or role, run the following commands:
   - aws iam get-user
   - aws iam get-role
2. (Optional) To view the users or roles on which a managed policy is used for a permissions boundary, run the following command:
   - aws iam list-entities-for-policy
3. (Optional) To view information about a managed policy, run the following commands:
   - To list managed policies: aws iam list-policies
   - To retrieve detailed information about a managed policy: aws iam get-policy
4. To remove a permissions boundary from a user or role, use one of the following commands:
   - aws iam delete-user-permissions-boundary
   - aws iam delete-role-permissions-boundary

To delete an inline policy (AWS CLI)

1. (Optional) To list all inline policies that are attached to an identity (user, user group, role), use one of the following commands:
   - aws iam list-user-policies
   - aws iam list-group-policies
   - aws iam list-role-policies
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, user group, or role), use one of the following commands:
   - aws iam get-user-policy
   - aws iam get-group-policy
   - aws iam get-role-policy
3. To delete an inline policy from an identity (user, user group, or role that is not a *service-linked role*), use one of the following commands:
   - aws iam delete-user-policy
   - aws iam delete-group-policy
   - aws iam delete-role-policy

Adding IAM policies (AWS API)

You can use the AWS API to attach managed policies that control permissions or specify a policy that serves as a permissions boundary. You can also embed an inline policy.

To use a managed policy as a permissions policy for an entity (AWS API)

1. (Optional) To view information about a policy, call the following operations:
   - To list managed policies: ListPolicies
   - To retrieve detailed information about a managed policy: GetPolicy
2. To attach a managed policy to an identity (user, user group, or role), call one of the following operations:
   - AttachUserPolicy
   - AttachGroupPolicy

- AttachRolePolicy

To use a managed policy to set a permissions boundary (AWS API)

1. (Optional) To view information about a managed policy, call the following operations:
   - To list managed policies: ListPolicies
   - To retrieve detailed information about a managed policy: GetPolicy
2. To use a managed policy to set the permissions boundary for an entity (user or role), call one of the following operations:
   - PutUserPermissionsBoundary
   - PutRolePermissionsBoundary

To embed an inline policy (AWS API)

To embed an inline policy in an identity (user, user group, or role that is not a *service-linked role*), call one of the following operations:

- PutUserPolicy
- PutGroupPolicy
- PutRolePolicy

Removing IAM policies (AWS API)

You can use the AWS API to detach managed policies that control permissions or remove a policy that serves as a permissions boundary. You can also delete an inline policy.

To detach a managed policy used as a permissions policy (AWS API)

1. (Optional) To view information about a policy, call the following operations:
   - To list managed policies: ListPolicies
   - To retrieve detailed information about a managed policy: GetPolicy
2. (Optional) To find out about the relationships between the policies and identities, call the following operations:

- To list the identities (IAM users, IAM groups, and IAM roles) to which a managed policy is attached:
  - ListEntitiesForPolicy
- To list the managed policies attached to an identity (a user, user group, or role), call one of the following operations:
  - ListAttachedUserPolicies
  - ListAttachedGroupPolicies
  - ListAttachedRolePolicies

3. To detach a managed policy from an identity (user, user group, or role), call one of the following operations:
   - DetachUserPolicy
   - DetachGroupPolicy
   - DetachRolePolicy

To remove a permissions boundary (AWS API)

1. (Optional) To view which managed policy is currently used to set the permissions boundary for a user or role, call the following operations:
   - GetUser
   - GetRole
2. (Optional) To view the users or roles on which a managed policy is used for a permissions boundary, call the following operation:
   - ListEntitiesForPolicy
3. (Optional) To view information about a managed policy, call the following operations:
   - To list managed policies: ListPolicies
   - To retrieve detailed information about a managed policy: GetPolicy
4. To remove a permissions boundary from a user or role, call one of the following operations:
   - DeleteUserPermissionsBoundary
   - DeleteRolePermissionsBoundary

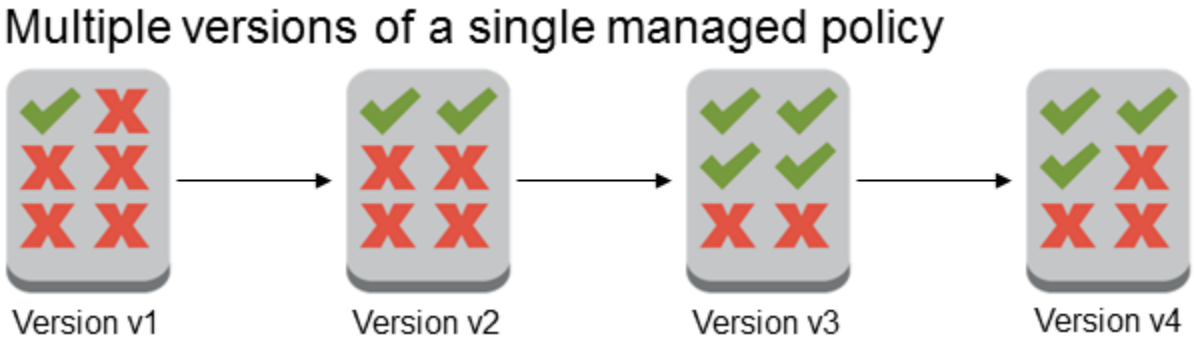
To delete an inline policy (AWS API)

1. (Optional) To list all inline policies that are attached to an identity (user, user group, role), call one of the following operations:
   - ListUserPolicies
   - ListGroupPolicies
   - ListRolePolicies
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, user group, or role), call one of the following operations:
   - GetUserPolicy
   - GetGroupPolicy
   - GetRolePolicy
3. To delete an inline policy from an identity (user, user group, or role that is not a *service-linked role*), call one of the following operations:
   - DeleteUserPolicy
   - DeleteGroupPolicy
   - DeleteRolePolicy

Versioning IAM policies

When you make changes to an IAM customer managed policy, and when AWS makes changes to an AWS managed policy, the changed policy doesn't overwrite the existing policy. Instead, IAM creates a new *version* of the managed policy. IAM stores up to five versions of your customer managed policies. IAM does not support versioning for inline policies.

The following diagram illustrates versioning for a customer managed policy. In this example, the versions 1-4 are saved. You can have up to five managed policy versions saved to IAM. When you edit a policy that would create a sixth saved version, you can choose which older version should no longer be saved. You can revert to any of the other four saved versions at any time.



A policy version is different from a `Version` policy element. The `Version` policy element is used within a policy and defines the version of the policy language. To learn more about the `Version` policy element see IAM JSON policy elements: Version.

You can use versions to track changes to a managed policy. For example, you might make a change to a managed policy and then discover that the change had unintended effects. In this case, you can roll back to a previous version of the managed policy by setting the previous version as the *default* version.

The following topics explain how you can use versioning for managed policies.

Topics

- Version limits
- Use versions to roll back changes
- Permissions for setting the default version of a policy
- Setting the default version of customer managed policies

Version limits

A managed policy can have up to five versions. If you need to make changes to a managed policy beyond five versions from the AWS Command Line Interface, or the AWS API, you must first delete one or more existing versions. If you use the AWS Management Console, you do not have to delete a version before editing your policy. When you save a sixth version, a dialog box appears that prompts you to delete one or more nondefault versions of your policy. You can view the JSON policy document for each version to help you decide. For details about this dialog box, see the section called “Edit IAM policies”.

You can delete any version of the managed policy that you want, except for the default version. When you delete a version, the version identifiers for the remaining versions do not change. As a

result, version identifiers might not be sequential. For example, if you delete versions v2 and v4 of a managed policy and add two new versions, the remaining version identifiers might be v1, v3, v5, v6, and v7.

Use versions to roll back changes

You can set the default version of a customer managed policy to roll back your changes. For example, consider the following scenario:

You create a customer managed policy that allows users to administer a particular Amazon S3 bucket using the AWS Management Console. Upon creation, your customer managed policy has only one version, identified as v1, so that version is automatically set as the default. The policy works as intended.

Later, you update the policy to add permission to administer a second Amazon S3 bucket. IAM creates a new version of the policy, identified as v2, that contains your changes. You set version v2 as the default, and a short time later your users report that they lack permission to use the Amazon S3 console. In this case, you can roll back to version v1 of the policy, which you know works as intended. To do this, you set version v1 as the default version. Your users are now able to use the Amazon S3 console to administer the original bucket.

Later, after you determine the error in version v2 of the policy, you update the policy again to add permission to administer the second Amazon S3 bucket. IAM creates another new version of the policy, identified as v3. You set version v3 as the default, and this version works as intended. At this point, you delete version v2 of the policy.

Permissions for setting the default version of a policy

The permissions that are required to set the default version of a policy correspond to the AWS API operations for the task. You can use the `CreatePolicyVersion` or `SetDefaultPolicyVersion` API operations to set the default version of a policy. To allow someone to set the default policy version of an existing policy, you can allow access to either the `iam:CreatePolicyVersion` action or the `iam:SetDefaultPolicyVersion` action. The `iam:CreatePolicyVersion` action allows them to create a new version of the policy and to set that version as the default. The `iam:SetDefaultPolicyVersion` action allows them to set any existing version of the policy as the default.

> **Important**
>
> To prevent a user from making changes to the default version of a policy, you must deny both `iam:CreatePolicyVersion` and `iam:SetDefaultPolicyVersion`.

You can use the following policy to deny a user access to change an existing customer managed policy:

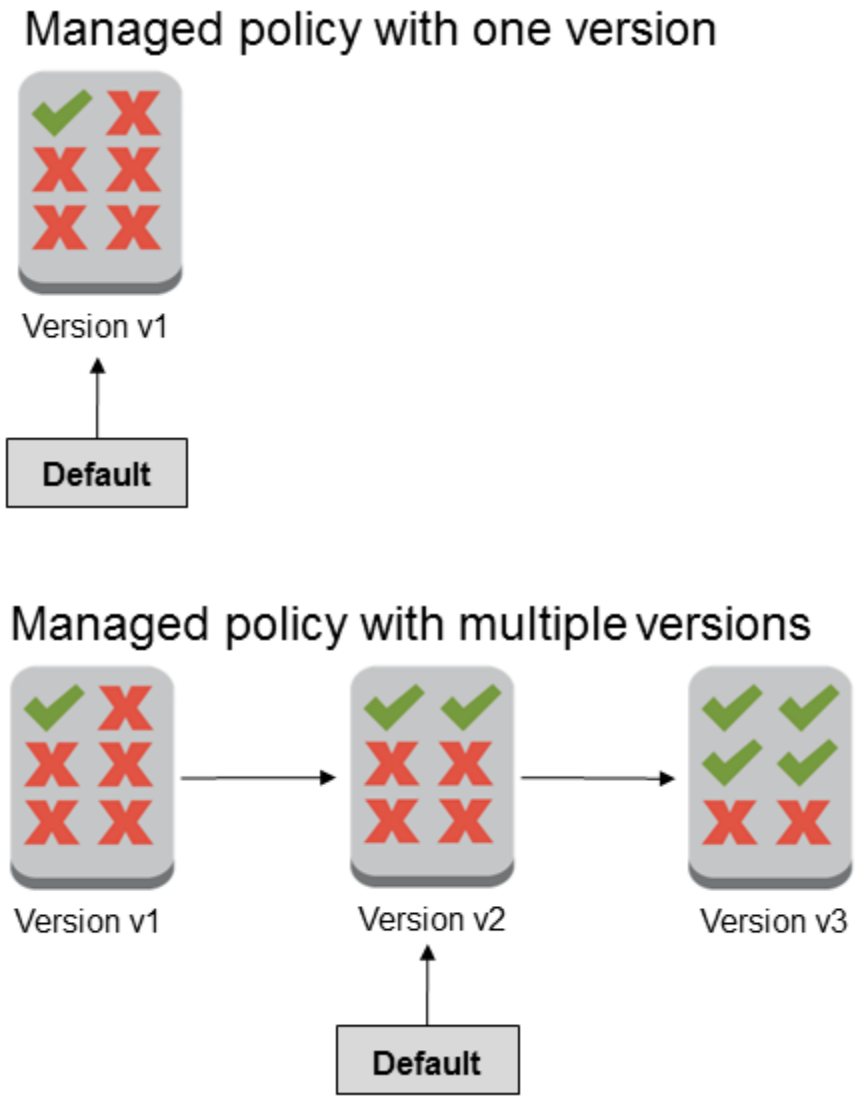
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iam:CreatePolicyVersion",
                "iam:SetDefaultPolicyVersion"
            ],
            "Resource": "arn:aws:iam::*:policy/POLICY-NAME"
        }
    ]
}
```

Setting the default version of customer managed policies

One of the versions of a managed policy is set as the *default* version. The policy's default version is the operative version—that is, it's the version that is in effect for all of the principal entities (IAM users, IAM groups, and IAM roles) that the managed policy is attached to.

When you create a customer managed policy, the policy begins with a single version identified as v1. For managed policies with only a single version, that version is automatically set as the default. For customer managed policies with more than one version, you choose which version to set as the default. For AWS managed policies, the default version is set by AWS. The following diagrams illustrate this concept.



You can set the default version of a customer managed policy to apply that version to every IAM identity (user, user group, and role) where the policy is attached. You cannot set the default version for an AWS managed policy or an inline policy.

To set the default version of a customer managed policy (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the policy name of the policy to set the default version of. You can use the search box to filter the list of policies.

4. Choose the **Policy versions** tab. Select the checkbox next to the version that you want to set as the default version, and then choose **Set as default**.

To learn how to set the default version of a customer managed policy from the AWS Command Line Interface or the AWS API, see Edit IAM policies (AWS CLI).

Edit IAM policies

A policy is an entity that, when attached to an identity or resource, defines their permissions. Policies are stored in AWS as JSON documents and are attached to principals as *identity-based policies* in IAM. You can attach an identity-based policy to a principal (or identity), such as an IAM user group, user, or role. Identity-based policies include AWS managed policies, customer managed policies, and inline policies. You can edit customer managed policies and inline policies in IAM. AWS managed policies cannot be edited. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

It's generally better to use customer managed policies instead of inline policies or AWS managed policies. AWS managed policies usually provide broad administrative or read-only permissions. Inline policies can't be reused on other identities or managed outside of the identity where they exist. For the greatest security, grant the least privilege, which means granting only the permissions required to perform specific job tasks.

When you create or edit IAM policies, AWS can automatically perform policy validation to help you create an effective policy with least privilege in mind. In the AWS Management Console, IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see IAM policy validation. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation.

You can use the AWS Management Console, AWS CLI, or AWS API to edit customer managed policies and inline policies in IAM. For more information about using CloudFormation templates to add or update policies, see AWS Identity and Access Management resource type reference in the *CloudFormation User Guide*.

Topics

- Edit IAM policies (console)
- Edit IAM policies (AWS CLI)
- Edit IAM policies (AWS API)

Edit IAM policies (console)

A policy is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS Management Console to edit *customer managed policies* and *inline policies* in IAM. AWS managed policies cannot be edited. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

For more information about policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

Prerequisites

Before you change the permissions for a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Editing customer managed policies (console)

You can edit customer managed policies to change the permissions that are defined in the policy from the AWS Management Console. A customer managed policy can have up to five versions. This is important because if you make changes to a managed policy beyond five versions, the AWS Management Console prompts you to decide which version to delete. You can also change the default version or delete a version of a policy before you edit it to avoid being prompted. To learn more about versions, see Versioning IAM policies.

Console

To edit a customer managed policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the policy name of the policy to edit. You can use the search box to filter the list of policies.
4. Choose the **Permissions** tab, and then choose **Edit**.
5. Do one of the following:

   - Choose the **Visual** option to change your policy without understanding JSON syntax. You can make changes to the service, actions, resources, or optional conditions for

each permission block in your policy. You can also import a policy to add additional permissions to the bottom of your policy. When you are finished making changes, choose **Next** to continue.

- Choose the **JSON** option to modify your policy by typing or pasting text in the JSON text box. You can also import a policy to add additional permissions to the bottom of your policy. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

6. On the **Review and save** page, review **Permissions defined in this policy** and then choose **Save changes** to save your work.
7. If the managed policy already has the maximum of five versions, choosing **Save changes** displays a dialog box. To save your new version, the oldest non-default version of the policy is removed and replaced with this new version. Optionally, you can set the new version as the default policy version.

   Choose **Save changes** to save your new policy version.

Setting the default version of a customer managed policy (console)

You can set a default version of a customer managed policy from the AWS Management Console. You can use this policy to establish a consistent baseline configuration for permissions across your organization. All new attachments of the policy will use this standardized set of permissions.

Console

To set the default version of a customer managed policy (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.

3. In the list of policies, choose the policy name of the policy to set the default version of. You can use the search box to filter the list of policies.
4. Choose the **Policy versions** tab. Select the check box next to the version that you want to set as the default version, and then choose **Set as default**.

Deleting a version of a customer managed policy (console)

You might need to delete a version of a customer managed policy to remove outdated or incorrect permissions that are no longer needed or pose potential security risks. By maintaining only necessary versions, you can help ensure that you stay within the limit of five managed policy versions, allowing room for future updates and refinements. You can delete a version of a customer managed policy from the AWS Management Console.

Console

To delete a version of a customer managed policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose the name of the customer managed policy that has a version you want to delete. You can use the search box to filter the list of policies.
4. Choose the **Policy versions** tab. Select the check box next to the version that you want to delete. Then choose **Delete**.
5. Confirm that you want to delete the version, and then choose **Delete**.

Editing inline policies (console)

You might need to edit a customer managed policy to update or refine the permissions granted, ensuring they remain aligned with your organization's evolving security requirements and access control needs. Editing allows you to adjust the policy's JSON document, adding, modifying, or removing specific actions, resources, or conditions to maintain the principle of least privilege and adapt to changes in your environment or processes. You can edit an inline policy from the AWS Management Console.

Console

To edit an inline policy for a user, user group, or role

1. In the navigation pane, choose **Users**, **User groups**, or **Roles**.
2. Choose the name of the user, user group, or role with the policy that you want to modify. Then choose the **Permissions** tab and expand the policy.
3. To edit an inline policy, choose **Edit Policy**.
4. Do one of the following:

   - Choose the **Visual** option to change your policy without understanding JSON syntax. You can make changes to the service, actions, resources, or optional conditions for each permission block in your policy. You can also import a policy to add additional permissions to the bottom of your policy. When you are finished making changes, choose **Next** to continue.
   - Choose the **JSON** option to modify your policy by typing or pasting text in the JSON text box. You can also import a policy to add additional permissions to the bottom of your policy. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**. To save your changes without affecting the currently attached entities, clear the check box for **Save as default version**.

   > **Note**
   >
   > You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

5. On the **Review** page, review the policy summary and then choose **Save changes** to save your work.

Edit IAM policies (AWS CLI)

A policy is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS Command Line Interface (AWS CLI) to edit *customer managed policies* and *inline*

policies in IAM. AWS managed policies cannot be edited. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

For more information about policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

Prerequisites

Before you change the permissions for a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Editing customer managed policies (AWS CLI)

You can edit a customer managed policy from the AWS CLI.

> **Note**
>
> A managed policy can have up to five versions. If you need to make changes to a customer managed policy beyond five versions, you must first delete one or more existing versions.

To edit a customer managed policy (AWS CLI)

1. (Optional) To view information about a policy, run the following commands:
   - To list managed policies: list-policies
   - To retrieve detailed information about a managed policy: get-policy
2. (Optional) To find out about the relationships between the policies and identities, run the following commands:
   - To list the identities (IAM users, IAM groups, and IAM roles) to which a managed policy is attached:
     - list-entities-for-policy
   - To list the managed policies attached to an identity (a user, user group, or role):
     - list-attached-user-policies
     - list-attached-group-policies

- list-attached-role-policies

3. To edit a customer managed policy, run the following command:

   - create-policy-version

4. (Optional) To validate a customer managed policy, run the following IAM Access Analyzer command:

   - validate-policy

Setting the default version of a customer managed policy (AWS CLI)

You can set a default version of a customer managed policy from the AWS CLI.

To set the default version of a customer managed policy (AWS CLI)

1. (Optional) To list managed policies, run the following command:

   - list-policies

2. To set the default version of a customer managed policy, run the following command:

   - set-default-policy-version

Deleting a version of a customer managed policy (AWS CLI)

You can delete a version of a customer managed policy from the AWS CLI.

To delete a version of a customer managed policy (AWS CLI)

1. (Optional) To list managed policies, run the following command:

   - list-policies

2. To delete a customer managed policy, run the following command:

   - delete-policy-version

Editing inline policies (AWS CLI)

You can edit an inline policy from the AWS CLI.

To edit an inline policy (AWS CLI)

1. (Optional) To view information about a policy, run the following commands:
   - To list inline policies associated to an identity (a user, user group, or role):
     - list-user-policies
     - list-role-policies
     - list-group-policies
   - To retrieve detailed information about an inline policy:
     - get-user-policy
     - get-role-policy
     - get-group-policy
2. To edit an inline policy, run the following command:
   - put-user-policy
   - put-role-policy
   - put-group-policy
3. (Optional) To validate an inline policy, run the following IAM Access Analyzer command:
   - validate-policy

Edit IAM policies (AWS API)

A policy is an entity that, when attached to an identity or resource, defines their permissions. You can use the AWS API to edit *customer managed policies* and *inline policies* in IAM. AWS managed policies cannot be edited. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

For more information about policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

Prerequisites

Before you change the permissions for a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Editing customer managed policies (AWS API)

You can edit a customer managed policy using the AWS API.

> **Note**
>
> A managed policy can have up to five versions. If you need to make changes to a customer managed policy beyond five versions, you must first delete one or more existing versions.

To edit a customer managed policy (AWS API)

1. (Optional) To view information about a policy, call the following operations:
   - To list managed policies: ListPolicies
   - To retrieve detailed information about a managed policy: GetPolicy
2. (Optional) To find out about the relationships between the policies and identities, call the following operations:
   - To list the identities (IAM users, IAM groups, and IAM roles) to which a managed policy is attached:
     - ListEntitiesForPolicy
   - To list the managed policies attached to an identity (a user, user group, or role):
     - ListAttachedUserPolicies
     - ListAttachedGroupPolicies
     - ListAttachedRolePolicies
3. To edit a customer managed policy, call the following operation:
   - CreatePolicyVersion
4. (Optional) To validate a customer managed policy, call the following IAM Access Analyzer operation:
   - ValidatePolicy

Setting the default version of a customer managed policy (AWS API)

You can set a default version of a customer managed policy from the AWS API.

To set the default version of a customer managed policy (AWS API)

1. (Optional) To list managed policies, call the following operation:
   - ListPolicies
2. To set the default version of a customer managed policy, call the following operation:
   - SetDefaultPolicyVersion

Deleting a version of a customer managed policy (AWS API)

You can delete a version of a customer managed policy from the AWS API.

To delete a version of a customer managed policy (AWS API)

1. (Optional) To list managed policies, call the following operation:
   - ListPolicies
2. To delete a customer managed policy, call the following operation:
   - DeletePolicyVersion

Editing inline policies (AWS API)

You can edit an inline policy from the AWS API.

To edit an inline policy (AWS API)

1. (Optional) To view information about an inline policy, call the following operations:
   - To list inline policies associated to an identity (a user, user group, or role):
     - ListUserPolicies
     - ListRolePolicies
     - ListGroupPolicies
   - To retrieve detailed information about an inline policy:
     - GetUserPolicy
     - GetRolePolicy
     - GetGroupPolicy

2. To edit an inline policy, call the following operations:

   - PutUserPolicy
   - PutRolePolicy
   - PutGroupPolicy

3. (Optional) To validate an inline policy, run the following IAM Access Analyzer operation:

   - ValidatePolicy

Delete IAM policies

You can delete IAM policies using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the AWS API.

> **Note**
>
> Deletion of IAM policies is permanent. After the policy is deleted it cannot be recovered.

For more information about IAM policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

For more information about the difference between managed and inline policies, see Managed policies and inline policies.

The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

Topics

- Delete IAM policies (console)
- Delete IAM policies (AWS CLI)
- Delete IAM policies (AWS API)

Delete IAM policies (console)

You can use the AWS Management Console to delete *customer managed policies* and *inline policies* in IAM. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

> **Note**
>
> Deletion of IAM policies is permanent. After the policy is deleted it cannot be recovered.

For more information about IAM policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

For more information about the difference between managed and inline policies, see Managed policies and inline policies.

Prerequisites

Before you delete a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Deleting IAM policies (console)

You might need to delete a customer managed policy when it becomes obsolete or no longer aligns with your organization's security requirements and access control needs. By deleting unnecessary policies, you reduce potential security risks associated with outdated or unused policies. You can delete a customer managed policy to remove it from your AWS account. You cannot delete AWS managed policies.

Console

To delete a customer managed policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Select the radio button next to the customer managed policy to delete. You can use the search box to filter the list of policies.
4. Choose **Actions**, and then choose **Delete**.
5. Follow the instructions to confirm that you want to delete the policy, and then choose **Delete**.

Deleting inline policies (console)

You might need to delete an inline policy when the specific permissions it grants are no longer required for the IAM user, group, or role to which it's directly attached. Deleting unnecessary inline policies helps reduce the risk of unintended access, especially since inline policies can't be reused or shared across multiple identities like managed policies can. You can delete an inline policy to remove it from your AWS account. You cannot delete AWS managed policies.

Console

To delete an inline policy for an IAM user, group, or role

1. In the navigation pane, choose **User groups**, **Users**, or **Roles**.
2. Choose the name of the user group, user, or role with the policy that you want to delete. Then choose the **Permissions** tab.
3. Select the checkboxes next to the policies to delete and choose **Remove**. Then, in the confirmation dialog, confirm the removal and deletion of the policy.
   - To delete an inline policy in **Users** or **Roles**, choose **Remove** to confirm the deletion.
   - If you are deleting a single inline policy in **User groups**, type the name of the policy and choose **Delete**. If you are deleting multiple inline policies in **User groups**, type the number of policies you are deleting followed by **`inline policies`** and choose **Delete**. For example, if you are deleting three inline policies, type **`3 inline policies`**.

Delete IAM policies (AWS CLI)

You can use the AWS Command Line Interface (AWS CLI) to delete *customer managed policies* and *inline policies* in IAM. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

> **Note**
>
> Deletion of IAM policies is permanent. After the policy is deleted it cannot be recovered.

For more information about IAM policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

For more information about the difference between managed and inline policies, see Managed policies and inline policies.

Prerequisites

Before you delete a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Deleting customer managed policies (AWS CLI)

You can delete a customer managed policy from the AWS Command Line Interface.

To delete a customer managed policy (AWS CLI)

1. (Optional) To view information about a policy, run the following commands:
   - To list managed policies: list-policies
   - To retrieve detailed information about a managed policy: get-policy
2. (Optional) To find out about the relationships between the policies and identities, run the following commands:
   - To list the identities (IAM users, IAM groups, and IAM roles) to which a managed policy is attached, run the following command:
     - list-entities-for-policy
   - To list the managed policies attached to an identity (a user, user group, or role), run one of the following commands:
     - list-attached-user-policies
     - list-attached-group-policies
     - list-attached-role-policies
3. To delete a customer managed policy, run the following command:
   - delete-policy

Deleting inline policies (AWS CLI)

You can delete an inline policy from the AWS CLI.

To delete an inline policy (AWS CLI)

1. (Optional) To list all inline policies that are attached to an identity (user, user group, role), use one of the following commands:
   - aws iam list-user-policies
   - aws iam list-group-policies
   - aws iam list-role-policies
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, user group, or role), use one of the following commands:
   - aws iam get-user-policy
   - aws iam get-group-policy
   - aws iam get-role-policy
3. To delete an inline policy from an identity (user, user group, or role that is not a *service-linked role*), use one of the following commands:
   - aws iam delete-user-policy
   - aws iam delete-group-policy
   - aws iam delete-role-policy

Delete IAM policies (AWS API)

You can use the AWS API to delete *customer managed policies* and *inline policies* in IAM. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

> **Note**
>
> Deletion of IAM policies is permanent. After the policy is deleted it cannot be recovered.

For more information about IAM policy structure and syntax, see Policies and permissions in AWS Identity and Access Management and the IAM JSON policy element reference.

For more information about the difference between managed and inline policies, see Managed policies and inline policies.

Prerequisites

Before you delete a policy, you should review its recent service-level activity. This is important because you don't want to remove access from a principal (person or application) who is using it. For more information about viewing last accessed information, see Refine permissions in AWS using last accessed information.

Deleting customer managed policies (AWS API)

You can delete a customer managed policy using the AWS API.

To delete a customer managed policy (AWS API)

1. (Optional) To view information about a policy, call the following operations:
   - To list managed policies: ListPolicies
   - To retrieve detailed information about a managed policy: GetPolicy
2. (Optional) To find out about the relationships between the policies and identities, call the following operations:
   - To list the identities (IAM users, IAM groups, and IAM roles) to which a managed policy is attached, call the following operation:
     - ListEntitiesForPolicy
   - To list the managed policies attached to an identity (a user, user group, or role), call one of the following operations:
     - ListAttachedUserPolicies
     - ListAttachedGroupPolicies
     - ListAttachedRolePolicies
3. To delete a customer managed policy, call the following operation:
   - DeletePolicy

Deleting inline policies (AWS API)

You can delete an inline policy using the AWS API.

To delete an inline policy (AWS API)

1. (Optional) To list all inline policies that are attached to an identity (user, user group, role), call one of the following operations:
   - ListUserPolicies
   - ListGroupPolicies
   - ListRolePolicies
2. (Optional) To retrieve an inline policy document that is embedded in an identity (user, user group, or role), call one of the following operations:
   - GetUserPolicy
   - GetGroupPolicy
   - GetRolePolicy
3. To delete an inline policy from an identity (user, user group, or role that is not a *service-linked role*), call one of the following operations:
   - DeleteUserPolicy
   - DeleteGroupPolicy
   - DeleteRolePolicy

Refine permissions in AWS using last accessed information

As an administrator, you might grant permissions to IAM resources (roles, users, user groups, or policies) beyond what they require. IAM provides last accessed information to help you identify unused permissions so that you can remove them. You can use last accessed information to refine your policies and allow access to only the services and actions that your IAM identities and policies use. This helps you to better adhere to the best practice of least privilege. You can view last accessed information for identities or policies that exist in IAM or AWS Organizations.

You can continuously monitor last accessed information with unused access analyzers. For more information, see Findings for external and unused access.

Topics

- Last accessed information types for IAM
- Last accessed information for AWS Organizations

- Things to know about last accessed information
- Permissions required
- Troubleshoot activity for IAM and AWS Organizations entities
- Where AWS tracks last accessed information
- View last accessed information for IAM
- View last accessed information for AWS Organizations
- Example scenarios for using last accessed information
- IAM action last accessed information services and actions

Last accessed information types for IAM

You can view two types of last accessed information for IAM identities: allowed AWS service information and allowed action information. The information includes the date and time when the attempt to access an AWS API was made. For actions, last accessed information reports service management actions. Management actions include creation, deletion, and modification actions. To learn more about how to view last accessed information for IAM, see View last accessed information for IAM.

For example scenarios for using last accessed information to make decisions about the permissions that you grant to your IAM identities, see Example scenarios for using last accessed information.

To learn more about how the information for management actions is provided, see Things to know about last accessed information.

Last accessed information for AWS Organizations

If you sign in using management account credentials, you can view service last accessed information for an AWS Organizations entity or policy in your organization. AWS Organizations entities include the organization root, organizational units (OUs), or accounts. Last accessed information for AWS Organizations includes information about services that are allowed by a service control policy (SCP). The information indicates which principals (root user, IAM user, or role) in an organization or account last attempted to access the service and when. To learn more about the report and how to view last accessed information for AWS Organizations, see View last accessed information for AWS Organizations.

For example scenarios for using last accessed information to make decisions about the permissions that you grant to your AWS Organizations entities, see Example scenarios for using last accessed information.

Things to know about last accessed information

Before you use last accessed information from a report to change the permissions for an IAM identity or AWS Organizations entity, review the following details about the information.

- **Tracking period** – Recent activity appears in the IAM console within four hours. The tracking period for service information is at least 400 days depending on when the service started tracking actions information. The tracking period for Amazon S3 actions information began on April, 12, 2020. The tracking period for Amazon EC2, IAM, and Lambda actions began on April 7, 2021. The tracking period for all other services began on May 23, 2023. For a list of services for which action last accessed information is available, see IAM action last accessed information services and actions. For more information on which Regions action last accessed information is available in, see Where AWS tracks last accessed information.
- **Attempts reported** – The service last accessed data includes all attempts to access an AWS API, not just the successful attempts. This includes all attempts that were made using the AWS Management Console, the AWS API through any of the SDKs, or any of the command line tools. An unexpected entry in the service last accessed data does not mean that your account has been compromised, because the request might have been denied. Refer to your CloudTrail logs as the authoritative source for information about all API calls and whether they were successful or denied access.
- **PassRole** – The `iam:PassRole` action is not tracked and is not included in IAM action last accessed information.
- **Action last accessed information** – Action last accessed information is available for service management actions accessed by IAM identities. See the list of services and their actions for which action last accessed reports information.

  > **Note**
  >
  > Action last accessed information is not available for any data plane event.

- **Management events** – IAM provides action information for service management events that are logged by CloudTrail. Sometimes, CloudTrail management events are also called control plane operations or control plane events. Management events provide visibility into

administrative operations that are performed on resources in your AWS account. To learn more about management events in CloudTrail, see Logging management events in the *AWS CloudTrail User Guide*.

- **Report owner** – Only the principal that generates a report can view the report details. This means that when you view the information in the AWS Management Console, you might have to wait for it to generate and load. If you use the AWS CLI or AWS API to get report details, your credentials must match the credentials of the principal that generated the report. If you use temporary credentials for a role or AWS STS federated user principal, you must generate and retrieve the report during the same session. For more information about assumed-role session principals, see AWS JSON policy elements: Principal.
- **IAM resources** – The last accessed information for IAM includes IAM resources (roles, users, IAM groups, and policies) in your account. Last accessed information for AWS Organizations includes principals (IAM users, IAM roles, or the AWS account root user) in the specified AWS Organizations entity. The last accessed information does not include unauthenticated attempts.
- **IAM policy types** – The last accessed information for IAM includes services that are allowed by an IAM identity's policies. These are policies attached to a role or attached to a user directly or through a group. Access allowed by other policy types is not included in your report. The excluded policy types include resource-based policies, access control lists, AWS Organizations SCPs, IAM permissions boundaries, and session policies. Permissions that are provided by service-linked roles are defined by the service that they are linked to and can't be modified in IAM. To learn more about service-linked roles, see Create a service-linked role To learn how the different policy types are evaluated to allow or deny access, see Policy evaluation logic.
- **AWS Organizations policy types** – The information for AWS Organizations includes only services that are allowed by an AWS Organizations entity's inherited service control policies (SCPs). SCPs are policies attached to a root, OU, or account. Access allowed by other policy types is not included in your report. The excluded policy types include identity-based policies, resource-based policies, access control lists, IAM permissions boundaries, and session policies. To learn how the different policy types are evaluated to allow or deny access, see Policy evaluation logic.
- **Specifying a policy ID** – When you use the AWS CLI or AWS API to generate a report for last accessed information in AWS Organizations, you can optionally specify a policy ID. The resulting report includes information for the services that are allowed by only that policy. The information includes the most recent account activity in the specified AWS Organizations entity or the entity's children. For more information, see aws iam generate-organizations-access-report or GenerateOrganizationsAccessReport.

- **AWS Organizations management account** – You must sign in to your organization's management account to view service last accessed information. You can choose to view information for the management account using the IAM console, the AWS CLI, or the AWS API. The resulting report lists all AWS services, because the management account is not limited by SCPs. If you specify a policy ID in the CLI or API, the policy is ignored. For each service, the report includes information for only the management account. However, reports for other AWS Organizations entities do not return information for activity in the management account.
- **AWS Organizations settings** – An administrator must enable SCPs in your organization root before you can generate data for AWS Organizations.

Permissions required

To view the last accessed information in the AWS Management Console, you must have a policy that grants the necessary permissions.

Permissions for IAM information

To use the IAM console to view the last accessed information for an IAM user, role, or policy, you must have a policy that includes the following actions:

- `iam:GenerateServiceLastAccessedDetails`
- `iam:Get*`
- `iam:List*`

These permissions allow a user to see the following:

- Which users, groups, or roles are attached to a managed policy
- Which services a user or role can access
- The last time they accessed the service
- The last time they attempted to use a specific Amazon EC2, IAM, Lambda, or Amazon S3 action

To use the AWS CLI or AWS API to view last accessed information for IAM, you must have permissions that match the operation you want to use:

- `iam:GenerateServiceLastAccessedDetails`
- `iam:GetServiceLastAccessedDetails`

- iam:GetServiceLastAccessedDetailsWithEntities
- iam:ListPoliciesGrantingServiceAccess

This example shows how you might create an identity-based policy that allows viewing IAM last accessed information. Additionally, it allows read-only access to all of IAM. This policy defines permissions for programmatic and console access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GenerateServiceLastAccessedDetails",
            "iam:Get*",
            "iam:List*"
        ],
        "Resource": "*"
    }
}
```

Permissions for AWS Organizations information

To use the IAM console to view a report for the root, OU, or account entities in AWS Organizations, you must have a policy that includes the following actions:

- iam:GenerateOrganizationsAccessReport
- iam:GetOrganizationsAccessReport
- organizations:DescribeAccount
- organizations:DescribeOrganization
- organizations:DescribeOrganizationalUnit
- organizations:DescribePolicy
- organizations:ListChildren
- organizations:ListParents
- organizations:ListPoliciesForTarget

- organizations:ListRoots
- organizations:ListTargetsForPolicy

To use the AWS CLI or AWS API to view service last accessed information for AWS Organizations, you must have a policy that includes the following actions:

- iam:GenerateOrganizationsAccessReport
- iam:GetOrganizationsAccessReport
- organizations:DescribePolicy
- organizations:ListChildren
- organizations:ListParents
- organizations:ListPoliciesForTarget
- organizations:ListRoots
- organizations:ListTargetsForPolicy

This example shows how you might create an identity-based policy that allows viewing service last accessed information for AWS Organizations. Additionally, it allows read-only access to all of AWS Organizations. This policy defines permissions for programmatic and console access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GenerateOrganizationsAccessReport",
            "iam:GetOrganizationsAccessReport",
            "organizations:Describe*",
            "organizations:List*"
        ],
        "Resource": "*"
    }
}
```

You can also use the iam:OrganizationsPolicyId condition key to allow generating a report only for a specific AWS Organizations policy. For an example policy, see IAM: View service last accessed information for an AWS Organizations policy.

Troubleshoot activity for IAM and AWS Organizations entities

In some cases, your AWS Management Console last accessed information table might be empty. Or perhaps your AWS CLI or AWS API request returns an empty set of information or a null field. In these cases, review the following issues:

- For action last accessed information, an action that you are expecting to see might not be returned in the list. This can happen either because the IAM identity does not have permissions for the action, or AWS does not yet track the action for last accessed information.
- For an IAM user, make sure that the user has at least one inline or managed policy attached, either directly or through group memberships.
- For an IAM group, verify that the group has at least one inline or managed policy attached.
- For an IAM group, the report returns only the service last accessed information for members that used the group's policies to access a service. To learn whether a member used other policies, review the last accessed information for that user.
- For an IAM role, verify that the role has at least one inline or managed policy attached.
- For an IAM entity (user or role), review other policy types that might affect the permissions of that entity. These include resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see Policy types or Policy evaluation for requests within a single account.
- For an IAM policy, make sure that the specified managed policy is attached to at least one user, group with members, or role.
- For an AWS Organizations entity (root, OU, or account), make sure that you are signed using AWS Organizations management account credentials.
- Verify that SCPs are enabled in your organization root.
- Action last accessed information is only available for the actions listed in IAM action last accessed information services and actions.

When you make changes, wait at least four hours for activity to appear in your IAM console report. If you use the AWS CLI or AWS API, you must generate a new report to view the updated information.

Where AWS tracks last accessed information

AWS collects last accessed information for the standard AWS Regions. When AWS adds additional Regions, those Regions are added to the following table, including the date that AWS started tracking information in each Region.

- **Service information** – The tracking period for services is at least 400 days, or less if your Region began tracking this feature within the last 400 days.
- **Actions information** – The tracking period for Amazon S3 management actions began on April, 12, 2020. The tracking period for Amazon EC2, IAM, and Lambda management actions began on April 7, 2021. The tracking period for management actions of all other services began on May 23, 2023. If a Region's tracking date is later than May 23, 2023, then action last accessed information from that Region will start at the later date.

Region name	Region	Tracking start date
US East (Ohio)	us-east-2	October 27, 2017
US East (N. Virginia)	us-east-1	October 1, 2015
US West (N. California)	us-west-1	October 1, 2015
US West (Oregon)	us-west-2	October 1, 2015
Africa (Cape Town)	af-south-1	April 22, 2020
Asia Pacific (Hong Kong)	ap-east-1	April 24, 2019
Asia Pacific (Hyderabad)	ap-south-2	November 22, 2022
Asia Pacific (Jakarta)	ap-southeast-3	December 13, 2021
Asia Pacific (Melbourne)	ap-southeast-4	January 23, 2023
Asia Pacific (Mumbai)	ap-south-1	June 27, 2016
Asia Pacific (Osaka)	ap-northeast-3	February 11, 2018
Asia Pacific (Seoul)	ap-northeast-2	January 6, 2016

Region name	Region	Tracking start date
Asia Pacific (Singapore)	ap-southeast-1	October 1, 2015
Asia Pacific (Sydney)	ap-southeast-2	October 1, 2015
Asia Pacific (Tokyo)	ap-northeast-1	October 1, 2015
Canada (Central)	ca-central-1	October 28, 2017
Europe (Frankfurt)	eu-central-1	October 1, 2015
Europe (Ireland)	eu-west-1	October 1, 2015
Europe (London)	eu-west-2	October 28, 2017
Europe (Milan)	eu-south-1	April 28, 2020
Europe (Paris)	eu-west-3	December 18, 2017
Europe (Spain)	eu-south-2	November 15, 2022
Europe (Stockholm)	eu-north-1	December 12, 2018
Europe (Zurich)	eu-central-2	November 8, 2022
Israel (Tel Aviv)	il-central-1	August 1, 2023
Middle East (Bahrain)	me-south-1	July 29, 2019
Middle East (UAE)	me-central-1	August 30, 2022
South America (São Paulo)	sa-east-1	December 11, 2015
AWS GovCloud (US-East)	us-gov-east-1	July 1, 2023
AWS GovCloud (US-West)	us-gov-west-1	July 1, 2023

If a Region is not listed in the previous table, then that Region does not yet provide last accessed information.

An AWS Region is a collection of AWS resources in a geographic area. Regions are grouped into partitions. The standard Regions are the Regions that belong to the `aws` partition. For more information about the different partitions, see Amazon Resource Names (ARNs) Format in the AWS General Reference. For more information about Regions, see About AWS Regions also in the AWS General Reference.

View last accessed information for IAM

You can view last accessed information for IAM using the AWS Management Console, AWS CLI, or AWS API. See the list of services and their actions for which last accessed information is displayed. For more information about last accessed information, see Refine permissions in AWS using last accessed information.

You can view information for the following resource types in IAM. In each case, the information includes allowed services for the given reporting period:

- **User** – View the last time that the user attempted to access each allowed service.
- **User group** – View information about the last time that a user group member attempted to access each allowed service. This report also includes the total number of members that attempted access.
- **Role** – View the last time that someone used the role in an attempt to access each allowed service.
- **Policy** – View information about the last time that a user or role attempted to access each allowed service. This report also includes the total number of entities that attempted access.

> **Note**
>
> Before you view the access information for a resource in IAM, make sure you understand the reporting period, reported entities, and the evaluated policy types for your information. For more details, see the section called “Things to know about last accessed information”.

Viewing information for IAM (console)

You can view last accessed information for IAM on the **Last Accessed** tab in the IAM console.

To view information for IAM (console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose either **User groups**, **Users**, **Roles**, or **Policies**.
3. Choose any user, user group, role, or policy name to open its **Summary** page and choose the **Last Accessed** tab. View the following information, based on the resource that you chose:

   - **User group** – View the list of services that user group members can access. You can also view when a member last accessed the service, what user group policies they used, and which user group member made the request. Choose the name of the policy to learn whether it is a managed policy or an inline user group policy. Choose the name of the user group member to see all of the members of the user group and when they last accessed the service.
   - **User** – View the list of services that the user can access. You can also view when they last accessed the service, and what policies are currently associated with the user. Choose the name of the policy to learn whether it is a managed policy, an inline user policy, or an inline policy for the user group.
   - **Role** – View the list of services that the role can access, when the role last accessed the service, and what policies were used. Choose the name of the policy to learn whether it is a managed policy or an inline role policy.
   - **Policy** – View the list of services with allowed actions in the policy. You can also view when the policy was last used to access the service, and which entity (user or role) used the policy. The **Last accessed** date also includes when access is granted to this policy through another policy. Choose the name of the entity to learn which entities have this policy attached and when they last accessed the service.
4. In the **Service** column of the table, choose the name of one of the services that includes action last accessed information to view a list of management actions that IAM entities have attempted to access. You can view the AWS Region and a timestamp that shows when someone last attempted to perform the action.
5. The **Last accessed** column is displayed for services and management actions of the services that include action last accessed information. Review the following possible results that are returned in this column. These results vary depending on whether a service or action is allowed, was accessed, and whether it is tracked by AWS for last accessed information.

<number of> days ago

The number of days since the service or action was used in the tracking period. The tracking period for services is for the last 400 days. The tracking period for Amazon S3 actions started on April 12, 2020. The tracking period for Amazon EC2, IAM, and Lambda actions started on April 7, 2021. The tracking period for all other services began on May 23, 2023. To learn more about the tracking start dates for each AWS Region, see Where AWS tracks last accessed information.

Not accessed in the tracking period

The tracked service or action has not been used by an entity in the tracking period.

It is possible for you to have permissions for an action that doesn't appear in the list. This can happen if the tracking information for the action is not currently included by AWS. You should not make permissions decisions based solely on the absence of tracking information. Instead, we recommend that you use this information to inform and support your overall strategy of granting least privilege. Check your policies to confirm that the level of access is appropriate.

Viewing information for IAM (AWS CLI)

You can use the AWS CLI to retrieve information about the last time that an IAM resource was used to attempt to access AWS services and Amazon S3, Amazon EC2, IAM, and Lambda actions. An IAM resource can be a user, user group, role, or policy.

To view information for IAM (AWS CLI)

1. Generate a report. The request must include the ARN of the IAM resource (user, user group, role, or policy) for which you want a report. You can specify the level of granularity that you want to generate in the report to view access details for either services or both services and actions. The request returns a `job-id` that you can then use in the `get-service-last-accessed-details` and `get-service-last-accessed-details-with-entities` operations to monitor the `job-status` until the job is complete.

   - aws iam generate-service-last-accessed-details

2. Retrieve details about the report using the `job-id` parameter from the previous step.

   - aws iam get-service-last-accessed-details

This operation returns the following information, based on the type of resource and level of granularity that you requested in the `generate-service-last-accessed-details` operation:

- **User** – Returns a list of services that the specified user can access. For each service, the operation returns the date and time of the user's last attempt and the ARN of the user.
- **User group** – Returns a list of services that members of the specified user group can access using the policies attached to the user group. For each service, the operation returns the date and time of the last attempt made by any user group member. It also returns the ARN of that user and the total number of user group members that have attempted to access the service. Use the GetServiceLastAccessedDetailsWithEntities operation to retrieve a list of all of the members.
- **Role** – Returns a list of services that the specified role can access. For each service, the operation returns the date and time of the role's last attempt and the ARN of the role.
- **Policy** – Returns a list of services for which the specified policy allows access. For each service, the operation returns the date and time that an entity (user or role) last attempted to access the service using the policy. It also returns the ARN of that entity and the total number of entities that attempted access.

3. Learn more about the entities that used user group or policy permissions in an attempt to access a specific service. This operation returns a list of entities with each entity's ARN, ID, name, path, type (user or role), and when they last attempted to access the service. You can also use this operation for users and roles, but it only returns information about that entity.

   - aws iam get-service-last-accessed-details-with-entities

4. Learn more about the identity-based policies that an identity (user, user group, or role) used in an attempt to access a specific service. When you specify an identity and service, this operation returns a list of permissions policies that the identity can use to access the specified service. This operation gives the current state of policies and does not depend on the generated report. It also does not return other policy types, such as resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see Policy types or Policy evaluation for requests within a single account.

   - aws iam list-policies-granting-service-access

Viewing information for IAM (AWS API)

You can use the AWS API to retrieve information about the last time that an IAM resource was used to attempt to access AWS services and Amazon S3, Amazon EC2, IAM, and Lambda actions. An IAM resource can be a user, user group, role, or policy. You can specify the level of granularity to generate in the report to view details for either services or both services and actions.

To view information for IAM (AWS API)

1. Generate a report. The request must include the ARN of the IAM resource (user, user group, role, or policy) for which you want a report. It returns a `JobId` that you can then use in the `GetServiceLastAccessedDetails` and `GetServiceLastAccessedDetailsWithEntities` operations to monitor the `JobStatus` until the job is complete.

   - GenerateServiceLastAccessedDetails

2. Retrieve details about the report using the `JobId` parameter from the previous step.

   - GetServiceLastAccessedDetails

   This operation returns the following information, based on the type of resource and level of granularity that you requested in the `GenerateServiceLastAccessedDetails` operation:

   - **User** – Returns a list of services that the specified user can access. For each service, the operation returns the date and time of the user's last attempt and the ARN of the user.
   - **User group** – Returns a list of services that members of the specified user group can access using the policies attached to the user group. For each service, the operation returns the date and time of the last attempt made by any user group member. It also returns the ARN of that user and the total number of user group members that have attempted to access the service. Use the GetServiceLastAccessedDetailsWithEntities operation to retrieve a list of all of the members.
   - **Role** – Returns a list of services that the specified role can access. For each service, the operation returns the date and time of the role's last attempt and the ARN of the role.
   - **Policy** – Returns a list of services for which the specified policy allows access. For each service, the operation returns the date and time that an entity (user or role) last attempted to access the service using the policy. It also returns the ARN of that entity and the total number of entities that attempted access.

3. Learn more about the entities that used user group or policy permissions in an attempt to access a specific service. This operation returns a list of entities with each entity's ARN, ID, name, path, type (user or role), and when they last attempted to access the service. You can also use this operation for users and roles, but it only returns information about that entity.

   - GetServiceLastAccessedDetailsWithEntities

4. Learn more about the identity-based policies that an identity (user, user group, or role) used in an attempt to access a specific service. When you specify an identity and service, this operation returns a list of permissions policies that the identity can use to access the specified service. This operation gives the current state of policies and does not depend on the generated report. It also does not return other policy types, such as resource-based policies, access control lists, AWS Organizations policies, IAM permissions boundaries, or session policies. For more information, see Policy types or Policy evaluation for requests within a single account.

   - ListPoliciesGrantingServiceAccess

View last accessed information for AWS Organizations

You can view service last accessed information for AWS Organizations using the IAM console, AWS CLI, or AWS API. For important information about the data, permissions required, troubleshooting, and supported Regions, see Refine permissions in AWS using last accessed information.

When you sign in to the IAM console using AWS Organizations management account credentials, you can view information for any entity in your organization. AWS Organizations entities include the organization root, organizational units (OUs), and accounts. You can also use the IAM console to view information for any service control policies (SCPs) in your organization. IAM shows a list of services that are allowed by any SCPs that apply to the entity. For each service, you can view the most recent account activity information for the chosen AWS Organizations entity or the entity's children.

When you use the AWS CLI or AWS API with management account credentials, you can generate a report for any entities or policies in your organization. A programmatic report for an entity includes a list of services that are allowed by any SCPs that apply to the entity. For each service, the report includes the most recent activity for accounts in the specified AWS Organizations entity or the entity's subtree.

When you generate a programmatic report for a policy, you must specify an AWS Organizations entity. This report includes a list of services that are allowed by the specified SCP. For each service,

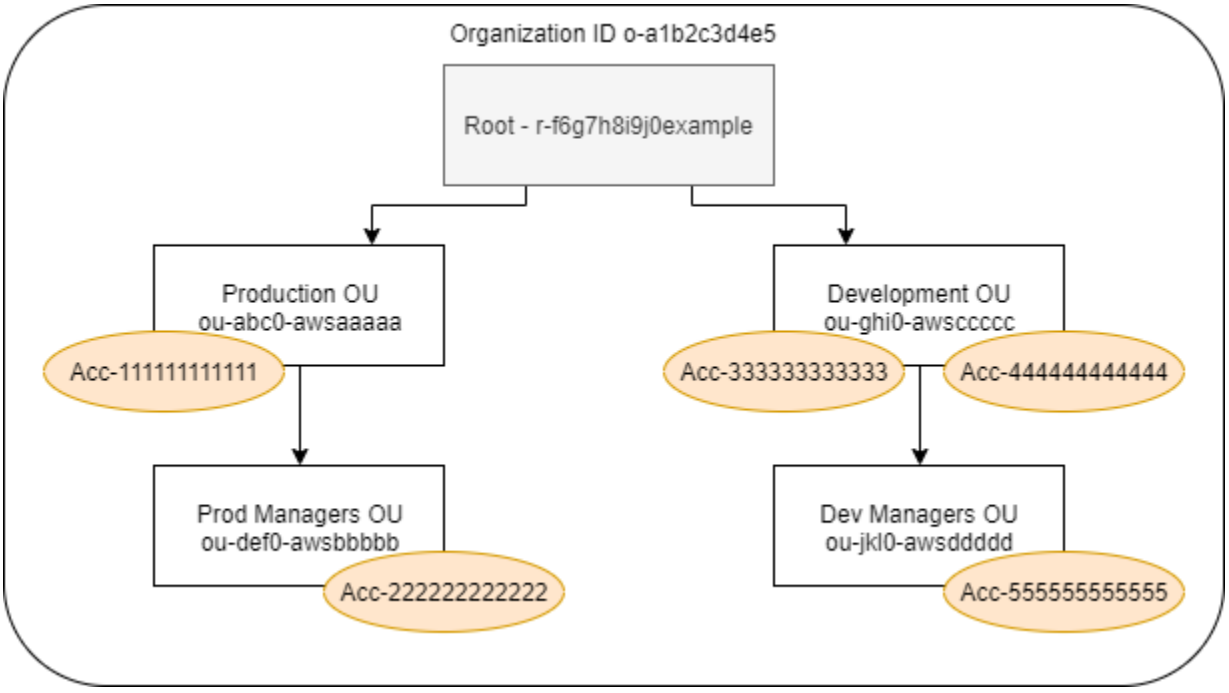
it includes the most recent account activity in the entity or entity's children that are granted permission by that policy. For more information, see aws iam generate-organizations-access-report or GenerateOrganizationsAccessReport.

Before you view the report, make sure that you understand the management account requirements and information, reporting period, reported entities, and the evaluated policy types. For more details, see the section called "Things to know about last accessed information".

Understand the AWS Organizations entity path

When you use the AWS CLI or AWS API to generate an AWS Organizations access report, you must specify an entity path. A path is a text representation of the structure of an AWS Organizations entity.

You can build an entity path using the known structure of your organization. For example, assume that you have the following organizational structure in AWS Organizations.



The path for the **Dev Managers** OU is built using the IDs of the organization, root, and all OUs in the path down to and including the OU.

```
o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-ghi0-awsccccc/ou-jkl0-awsdddddd/
```

The path for the account in the **Production** OU is built using the IDs of the organization, root, the OU, and the account number.

```
o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-abc0-awsaaaaa/111111111111/
```

> **Note**
>
> Organization IDs are globally unique but OU IDs and root IDs are unique only within an organization. This means that no two organizations share the same organization ID. However, another organization might have an OU or root with the same ID as yours. We recommend that you always include the organization ID when you specify an OU or root.

Viewing information for AWS Organizations (console)

You can use the IAM console to view service last accessed information for your root, OU, account, or policy.

To view information for the root (console)

1. Sign in to the AWS Management Console using AWS Organizations management account credentials, and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane below the **Access reports** section, choose **Organization activity**.
3. On the **Organization activity** page, choose **Root**.
4. On the **Details and activity** tab, view the **Service access report** section. The information includes a list of services that are allowed by the policies that are attached directly to the root. The information shows you from which account the service was last accessed and when. For more details about which principal accessed the service, sign in as an administrator in that account and view the IAM service last accessed information.
5. Choose the **Attached SCPs** tab to view the list of the service control policies (SCPs) that are attached to the root. IAM shows you the number of target entities to which each policy is attached. You can use this information to decide which SCP to review.
6. Choose the name of an SCP to view all of the services that the policy allows. For each service, view from which account the service was last accessed, and when.
7. Choose **Edit in AWS Organizations** to view additional details and edit the SCP in the AWS Organizations console. For more information, see Updating an SCP in the *AWS Organizations User Guide*.

To view information for an OU or account (console)

1. Sign in to the AWS Management Console using AWS Organizations management account credentials, and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane below the **Access reports** section, choose **Organization activity**.
3. On the **Organization activity** page, expand the structure of your organization. Then choose the name of the OU or any account that you want to view except the management account.
4. On the **Details and activity** tab, view the **Service access report** section. The information includes a list of services that are allowed by the SCPs attached to the OU or account *and* all of its parents. The information shows you from which account the service was last accessed and when. For more details about which principal accessed the service, sign in as an administrator in that account and view the IAM service last accessed information.
5. Choose the **Attached SCPs** tab to view the list of the service control policies (SCPs) that are attached directly to the OU or account. IAM shows you the number of target entities to which each policy is attached. You can use this information to decide which SCP to review.
6. Choose the name of an SCP to view all of the services that the policy allows. For each service, view from which account the service was last accessed, and when.
7. Choose **Edit in AWS Organizations** to view additional details and edit the SCP in the AWS Organizations console. For more information, see Updating an SCP in the *AWS Organizations User Guide*.

To view information for the management account (console)

1. Sign in to the AWS Management Console using AWS Organizations management account credentials, and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane below the **Access reports** section, choose **Organization activity**.
3. On the **Organization activity** page, expand the structure of your organization and choose the name your management account.
4. On the **Details and activity** tab, view the **Service access report** section. The information includes a list of all AWS services. The management account is not limited by SCPs. The information shows you whether the account last accessed the service and when. For more details about which principal accessed the service, sign in as an administrator in that account and view the IAM service last accessed information.
5. Choose the **Attached SCPs** tab to confirm that there are no attached SCPs because the account is the management account.

To view information for a policy (console)

1. Sign in to the AWS Management Console using AWS Organizations management account credentials, and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane below the **Access reports** section, choose **Service control policies (SCPs)**.
3. On the **Service control policies (SCPs)** page, view a list of the policies in your organization. You can view the number of target entities to which each policy is attached.
4. Choose the name of an SCP to view all of the services that the policy allows. For each service, view from which account the service was last accessed, and when.
5. Choose **Edit in AWS Organizations** to view additional details and edit the SCP in the AWS Organizations console. For more information, see Updating an SCP in the *AWS Organizations User Guide*.

Viewing information for AWS Organizations (AWS CLI)

You can use the AWS CLI to retrieve service last accessed information for your AWS Organizations root, OU, account, or policy.

To view AWS Organizations service last accessed information (AWS CLI)

1. Use your AWS Organizations management account credentials with the required IAM and AWS Organizations permissions, and confirm that SCPs are enabled for your root. For more information, see Things to know about last accessed information.
2. Generate a report. The request must include the path of the AWS Organizations entity (root, OU, or account) for which you want a report. You can optionally include an `organization-policy-id` parameter to view a report for a specific policy. The command returns a `job-id` that you can then use in the `get-organizations-access-report` command to monitor the `job-status` until the job is complete.

   - aws iam generate-organizations-access-report
3. Retrieve details about the report using the `job-id` parameter from the previous step.

   - aws iam get-organizations-access-report

   This command returns a list of services that entity members can access. For each service, the command returns the date and time of an account member's last attempt and the entity path

of the account. It also returns the total number of services that are available to access and the number of services that were not accessed. If you specified the optional `organizations-policy-id` parameter, then the services that are available to access are those that are allowed by the specified policy.

Viewing information for AWS Organizations (AWS API)

You can use the AWS API to retrieve service last accessed information for your AWS Organizations root, OU, account, or policy.

To view AWS Organizations service last accessed information (AWS API)

1. Use your AWS Organizations management account credentials with the required IAM and AWS Organizations permissions, and confirm that SCPs are enabled for your root. For more information, see Things to know about last accessed information.
2. Generate a report. The request must include the path of the AWS Organizations entity (root, OU, or account) for which you want a report. You can optionally include an `OrganizationsPolicyId` parameter to view a report for a specific policy. The operation returns a `JobId` that you can then use in the `GetOrganizationsAccessReport` operation to monitor the `JobStatus` until the job is complete.

   - GenerateOrganizationsAccessReport
3. Retrieve details about the report using the `JobId` parameter from the previous step.

   - GetOrganizationsAccessReport

   This operation returns a list of services that entity members can access. For each service, the operation returns the date and time of an account member's last attempt and the entity path of the account. It also returns the total number of services that are available to access, and the number of services that were not accessed. If you specified the optional `OrganizationsPolicyId` parameter, then the services that are available to access are those that are allowed by the specified policy.

Example scenarios for using last accessed information

You can use last accessed information to make decisions about the permissions that you grant to your IAM entities or AWS Organizations entities. For more information, see Refine permissions in AWS using last accessed information.

> **Note**
>
> Before you view the access information for an entity or policy in IAM or AWS Organizations, make sure that you understand the reporting period, reported entities, and the evaluated policy types for your data. For more details, see the section called "Things to know about last accessed information".

It's up to you as an administrator to balance the accessibility and least privilege that's appropriate for your company.

Using information to reduce permissions for an IAM group

You can use last accessed information to reduce IAM group permissions to include only those services that your users need. This method is an important step in granting least privilege at a service level.

For example, Paulo Santos is the administrator in charge of defining AWS user permissions for Example Corp. This company just started using AWS, and the software development team has not yet defined what AWS services they will use. Paulo wants to give the team permission to access only the services they need, but since that is not yet defined, he temporarily gives them power-user permissions. Then he uses last accessed information to reduce the group's permissions.

Paulo creates a managed policy named `ExampleDevelopment` using the following JSON text. He then attaches it to a group named `Development` and adds all of the developers to the group.

> **Note**
>
> Paulo's power users might need `iam:CreateServiceLinkedRole` permissions to use some services and features. He understands that adding this permission allows the users to create any service-linked role. He accepts this risk for his power users.

JSON

```
{

    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessToAllServicesExceptPeopleManagement",
            "Effect": "Allow",
            "NotAction": [
                "iam:*",
                "organizations:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RequiredIamAndOrgsActions",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole",
                "iam:ListRoles",
                "organizations:DescribeOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

Paulo decides to wait for 90 days before he views the last accessed information for the `Development` group using the AWS Management Console. He views the list of services that the group members accessed. He learns that the users accessed five services within the last week: AWS CloudTrail, Amazon CloudWatch Logs, Amazon EC2, AWS KMS, and Amazon S3. They accessed a few other services when they were first evaluating AWS, but not since then.

Paulo decides to reduce the policy permissions to include only those five services and the required IAM and AWS Organizations actions. He edits `ExampleDevelopment` policy using the following JSON text.

> **Note**
>
> Paulo's power users might need `iam:CreateServiceLinkedRole` permissions to use some services and features. He understands that adding this permission allows the users to create any service-linked role. He accepts this risk for his power users.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessToListedServices",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "kms:*",
                "cloudtrail:*",
                "logs:*",
                "ec2:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RequiredIamAndOrgsActions",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole",
                "iam:ListRoles",
                "organizations:DescribeOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

To further reduce permissions, Paulo can view the account's events in AWS CloudTrail **Event history**. There he can view detailed event information that he can use to reduce the policy's permissions to include only the actions and resources that the developers need. For more

information, see Viewing CloudTrail Events in the CloudTrail Console in the *AWS CloudTrail User Guide*.

Using information to reduce permissions for an IAM user

You can use last accessed information to reduce the permissions for an individual IAM user.

For example, Martha Rivera is an IT administrator responsible for ensuring that people in her company do not have excess AWS permissions. As part of a periodic security check, she reviews the permissions of all IAM users. One of these users is an application developer named Nikhil Jayashankar, who previously filled the role of a security engineer. Because of the change in job requirements, Nikhil is a member of both the `app-dev` group and the `security-team` group. The `app-dev` group for his new job grants permissions to multiple services including Amazon EC2, Amazon EBS, Auto Scaling, Amazon S3, Route 53, and Elastic Transcoder. The `security-team` group for his old job grants permissions to IAM and CloudTrail.

As an administrator, Martha signs into the IAM console and chooses **Users**, chooses the name `nikhilj`, and then chooses the **Last Accessed** tab.

Martha reviews the **Last Accessed** column and notices that Nikhil has not recently accessed IAM, CloudTrail, Route 53, Amazon Elastic Transcoder, and a number of other AWS services. Nikhil has accessed Amazon S3. Martha chooses **S3** from the list of services and learns that Nikhil has performed some Amazon S3 `List` actions in the last two weeks. Within her company, Martha confirms that Nikhil has no business need to access IAM and CloudTrail anymore because he is no longer a member of the internal security team.

Martha is now ready to act on the service and action last accessed information. However, unlike the group in the previous example, an IAM user like `nikhilj` might be subject to multiple policies and be a member of multiple groups. Martha must proceed with caution to avoid inadvertently disrupting access for `nikhilj` or other group members. In addition to learning what access Nikhil should have, she must determine *how* he is receiving these permissions.

Martha chooses the **Permissions** tab, where she views which policies are attached directly to `nikhilj` and those attached from a group. She expands each policy and views the policy summary to learn which policy allows access to the services that Nikhil is not using:

- IAM – The `IAMFullAccess` AWS managed policy is attached directly to `nikhilj` and attached to the `security-team` group.
- CloudTrail – The `AWSCloudTrailReadOnlyAccess` AWS managed policy is attached to the `security-team` group.

- Route 53 – The `App-Dev-Route53` customer managed policy is attached to the `app-dev` group.
- Elastic Transcoder – The `App-Dev-ElasticTranscoder` customer managed policy is attached to the `app-dev` group.

Martha decides to remove the `IAMFullAccess` AWS managed policy that is attached directly to `nikhilj`. She also removes Nikhil's membership to the `security-team` group. These two actions remove the unnecessary access to IAM and CloudTrail.

Nikhil's permissions to access to Route 53 and Elastic Transcoder are granted by the `app-dev` group. Although Nikhil isn't using those services, other members of the group might be. Martha reviews the last accessed information for the `app-dev` group and learns that several members recently accessed Route 53 and Amazon S3. But no group members have accessed Elastic Transcoder in the last year. She removes the `App-Dev-ElasticTranscoder` customer managed policy from the group.

Martha then reviews the last accessed information for the `App-Dev-ElasticTranscoder` customer managed policy. She learns that the policy is not attached to any other IAM identities. She investigates within her company to make sure that the policy will not be needed in the future, and then she deletes it.

Using information before deleting IAM resources

You can use last accessed information before you delete an IAM resource to make sure that a certain amount of time has passed since someone last used the resource. This applies to users, groups, roles, and policies. To learn more about these actions, see the following topics:

- **IAM users** – Remove or deactivate an IAM user
- **Groups** – Delete an IAM group
- **Roles** – Delete roles or instance profiles
- **Policies** – Delete IAM policies (this also detaches the policy from identities)

Using information before editing IAM policies

You can review last accessed information for an IAM identity (user, group, or role), or for an IAM policy before editing a policy that affects that resource. This is important because you don't want to remove access for someone that is using it.

For example, Arnav Desai is a developer and AWS administrator for Example Corp. When his team started using AWS, they gave all developers power-user access that allowed them full access to all services except IAM and AWS Organizations. As a first step towards granting least privilege, Arnav wants to use the AWS CLI to review the managed policies in his account.

To do this, Arnav first lists the customer managed permissions policies in his account that are attached to an identity, using the following command:

```
aws iam list-policies --scope Local --only-attached --policy-usage-filter
 PermissionsPolicy
```

From the response, he captures the ARN for each policy. Arnav then generates a report for last accessed information for each policy using the following command.

```
aws iam generate-service-last-accessed-details --arn arn:aws:iam::123456789012:policy/
ExamplePolicy1
```

From that response, he captures the ID of the generated report from the `JobId` field. Arnav then polls the following command until the `JobStatus` field returns a value of `COMPLETED` or `FAILED`. If the job failed, he captures the error.

```
aws iam get-service-last-accessed-details --job-id 98a765b4-3cde-2101-2345-example678f9
```

When the job has a status of `COMPLETED`, Arnav parses the contents of the JSON-formatted `ServicesLastAccessed` array.

```
 "ServicesLastAccessed": [
        {
            "TotalAuthenticatedEntities": 1,
            "LastAuthenticated": 2018-11-01T21:24:33.222Z,
            "ServiceNamespace": "dynamodb",
            "LastAuthenticatedEntity": "arn:aws:iam::123456789012:user/IAMExampleUser",
            "ServiceName": "Amazon DynamoDB"
        },

        {
            "TotalAuthenticatedEntities": 0,
            "ServiceNamespace": "ec2",
            "ServiceName": "Amazon EC2"
        },
```

```
        {
            "TotalAuthenticatedEntities": 3,
            "LastAuthenticated": 2018-08-25T15:29:51.156Z,
            "ServiceNamespace": "s3",
            "LastAuthenticatedEntity": "arn:aws:iam::123456789012:role/IAMExampleRole",
            "ServiceName": "Amazon S3"
        }
    ]
```

From this information, Arnav learns that the `ExamplePolicy1` policy allows access to three services, Amazon DynamoDB, Amazon S3, and Amazon EC2. The IAM user named `IAMExampleUser` last attempted to access DynamoDB on November 1, and someone used the `IAMExampleRole` role to attempt to access Amazon S3 on August 25. There are also two more entities that attempted to access Amazon S3 in the last year. However, nobody has attempted to access Amazon EC2 in the last year.

This means that Arnav can safely remove the Amazon EC2 actions from the policy. Arnav wants to review the current JSON document for the policy. First, he must determine the version number of the policy using the following command.

```
aws iam list-policy-versions --policy-arn arn:aws:iam::123456789012:policy/
ExamplePolicy1
```

From the response, Arnav collects the current default version number from the `Versions` array. He then uses that version number (`v2`) to request the JSON policy document using the following command.

```
aws iam get-policy-version --policy-arn arn:aws:iam::123456789012:policy/ExamplePolicy1
 --version-id v2
```

Arnav stores the JSON policy document returned in the `Document` field of the `PolicyVersion` array. Within the policy document, Arnav searches for actions with in the `ec2` namespace. If there are no actions from other namespaces remaining in the policy, then he detaches the policy from the affected identities (users, groups, and roles). He then deletes the policy. In this case, the policy does include the Amazon DynamoDB and Amazon S3 services. So Arnav removes the Amazon EC2 actions from the document and saves his changes. He then uses the following command to update the policy using the new version of the document and to set that version as the default policy version.

```
aws iam create-policy-version --policy-arn arn:aws:iam::123456789012:policy/
ExamplePolicy1 --policy-document file://UpdatedPolicy.json --set-as-default
```

The `ExamplePolicy1` policy is now updated to remove access to the unnecessary Amazon EC2 service.

Other IAM scenarios

Information about when an IAM resource (user, group, role, or policy) last attempted to access a service can help you when you complete any of the following tasks:

- **Policies** – Editing an existing customer-managed or inline policy to remove permissions
- **Policies** – Converting an inline policy to a managed policy and then deleting it
- **Policies** – Adding an explicit deny to an existing policy
- **Policies** – Detaching a managed policy from an identity (user, group, or role)
- **Entities** – Set a permissions boundary to control the maximum permissions that an entity (user or role) can have
- **Groups** – Removing users from a group

Using information to refine permissions for an organizational unit

You can use last accessed information to refine the permissions for an organizational unit (OU) in AWS Organizations.

For example, John Stiles is an AWS Organizations administrator. He is responsible for ensuring that people in company AWS accounts do not have excess permissions. As part of a periodic security audit, he reviews the permissions of his organization. His `Development` OU contains accounts that are often used to test new AWS services. John decides to periodically review the report for services that have not been accessed in more than 180 days. He then removes permissions for the OU members to access those services.

John signs into the IAM console using his management account credentials. In the IAM console, he locates the AWS Organizations data for the `Development` OU. He reviews the **Service access report** table and sees two AWS services that have not been accessed in more than his preferred period of 180 days. He remembers adding permissions for the development teams to access Amazon Lex and AWS Database Migration Service. John contacts the development teams and confirms that they no longer have a business need to test these services.

John is now ready to act on the last accessed information. He chooses **Edit in AWS Organizations** and is reminded that the SCP is attached to multiple entities. He chooses **Continue**. In AWS Organizations, he reviews the targets to learn to which AWS Organizations entities that the SCP is attached. All of entities are within the `Development` OU.

John decides to deny access to the Amazon Lex and AWS Database Migration Service actions in the `NewServiceTest` SCP. This action removes the unnecessary access to the services.

IAM action last accessed information services and actions

The following table lists the AWS services for which IAM action last accessed information is displayed. For a list of actions in each service, see Actions, resources, and condition keys for AWS services in the Service Authorization Reference.

AWS provides action last accessed information in JSON format to streamline the automation of policy management workflows. With the service reference information, you can access action last accessed information across AWS services from machine-readable files. For more information, see Simplified AWS service information for programmatic access in the Service Authorization Reference.

Service	Service prefix
AWS Identity and Access Management and Access Analyzer	access-analyzer
AWS Account Management	account
AWS Certificate Manager	acm
Amazon Managed Workflows for Apache Airflow	airflow
AWS Amplify	amplify
AWS Amplify UI Builder	amplifyuibuilder
Amazon AppIntegrations	app-integrations
AWS AppConfig	appconfig
Amazon AppFlow	appflow

Service	Service prefix
AWS Application Cost Profiler	application-cost-profiler
Amazon CloudWatch Application Insights	applicationinsights
AWS App Mesh	appmesh
Amazon WorkSpaces Applications	appstream
AWS AppSync	appsync
Amazon Managed Service for Prometheus	aps
Amazon Athena	athena
AWS Audit Manager	auditmanager
AWS Auto Scaling	autoscaling
AWS Marketplace	aws-marketplace
AWS Backup	backup
AWS Batch	batch
Amazon Braket	braket
AWS Budgets	budgets
AWS Cloud9	cloud9
AWS CloudFormation	cloudformation
Amazon CloudFront	cloudfront
AWS CloudHSM	cloudhsm
Amazon CloudSearch	cloudsearch
AWS CloudTrail	cloudtrail

Service	Service prefix
Amazon CloudWatch	cloudwatch
AWS CodeArtifact	codeartifact
AWS CodeDeploy	codedeploy
Amazon CodeGuru Profiler	codeguru-profiler
Amazon CodeGuru Reviewer	codeguru-reviewer
AWS CodePipeline	codepipeline
AWS CodeStar	codestar
AWS CodeStar Notifications	codestar-notifications
Amazon Cognito Identity	cognito-identity
Amazon Cognito user pools	cognito-idp
Amazon Cognito Sync	cognito-sync
Amazon Comprehend Medical	comprehendmedical
AWS Compute Optimizer	compute-optimizer
AWS Config	config
Amazon Connect Customer	connect
AWS Cost and Usage Report	cur
AWS Glue DataBrew	databrew
AWS Data Exchange	dataexchange
AWS Data Pipeline	datapipeline

Service	Service prefix
DynamoDB Accelerator	dax
AWS Device Farm	devicefarm
Amazon DevOps Guru	devops-guru
AWS Direct Connect	directconnect
Amazon Data Lifecycle Manager	dlm
AWS Database Migration Service	dms
Amazon DocumentDB Elastic Clusters	docdb-elastic
Amazon DynamoDB	dynamodb
Amazon Elastic Block Store	ebs
Amazon Elastic Compute Cloud	ec2
Amazon Elastic Container Registry	ecr
Amazon Elastic Container Registry Public	ecr-public
Amazon Elastic Container Service	ecs
Amazon Elastic Kubernetes Service	eks
Amazon ElastiCache	elasticache
AWS Elastic Beanstalk	elasticbeanstalk
Amazon Elastic File System	elasticfilesystem
Elastic Load Balancing	elasticloadbalancing
Amazon EMR on EKS (EMR Containers)	emr-containers
Amazon EMR Serverless	emr-serverless

Service	Service prefix
Amazon OpenSearch Service	es
Amazon EventBridge	events
Amazon FinSpace	finspace
Amazon Data Firehose	firehose
AWS Fault Injection Service	fis
AWS Firewall Manager	fms
Amazon Fraud Detector	frauddetector
Amazon FSx	fsx
Amazon GameLift Servers	gamelift
Amazon Location Service	geo
Amazon Glacier	glacier
Amazon Managed Grafana	grafana
AWS IoT Greengrass	greengrass
AWS Ground Station	groundstation
Amazon GuardDuty	guardduty
AWS HealthLake	healthlake
Amazon Honeycode	honeycode
AWS Identity and Access Management	iam
AWS Identity Store	identitystore
EC2 Image Builder	imagebuilder

Service	Service prefix
Amazon Inspector Classic	inspector
Amazon Inspector	inspector2
AWS IoT	iot
AWS IoT Analytics	iotanalytics
AWS IoT Core Device Advisor	iotdeviceadvisor
AWS IoT Events	iotevents
AWS IoT Fleet Hub	iotfleethub
AWS IoT SiteWise	iotsitewise
AWS IoT TwinMaker	iottwinmaker
AWS IoT Wireless	iotwireless
Amazon Interactive Video Service	ivs
Amazon Interactive Video Service Chat	ivschat
Amazon Managed Streaming for Apache Kafka	kafka
Amazon Managed Streaming for Kafka Connect	kafkaconnect
Amazon Kendra	kendra
Amazon Kinesis	kinesis
Amazon Kinesis Analytics V2	kinesisanalytics
AWS Key Management Service	kms
AWS Lambda	lambda
Amazon Lex V2	lex

Service	Service prefix
AWS License Manager Linux Subscriptions Manager	license-manager-linux-subscriptions
Amazon Lightsail	lightsail
Amazon CloudWatch Logs	logs
Amazon Lookout for Equipment	lookoutequipment
Amazon Lookout for Metrics	lookoutmetrics
AWS Mainframe Modernization	m2
Amazon Managed Blockchain	managedblockchain
AWS Elemental MediaConnect	mediaconnect
AWS Elemental MediaConvert	mediaconvert
AWS Elemental MediaLive	medialive
AWS Elemental MediaTailor	mediatailor
Amazon MemoryDB	memorydb
AWS Transform MGN	mgn
AWS Migration Hub	mgh
AWS Migration Hub Strategy Recommendations	migrationhub-strategy
Amazon Pinpoint	mobiletargeting
Amazon MQ	mq
AWS Network Manager	networkmanager
Amazon Nimble Studio	nimble

Service	Service prefix
AWS HealthOmics	omics
AWS OpsWorks	opsworks
AWS OpsWorks CM	opsworks-cm
AWS Outposts	outposts
AWS Organizations	organizations
AWS Performance Insights	pi
Amazon EventBridge Pipes	pipes
Amazon Polly	polly
Amazon Connect Customer Customer Profiles	profile
Amazon QLDB	qldb
AWS Resource Access Manager	ram
AWS Recycle Bin	rbin
Amazon Relational Database Service	rds
Amazon Redshift	redshift
Amazon Redshift Data API	redshift-data
AWS Migration Hub Refactor Spaces	refactor-spaces
Amazon Rekognition	rekognition
AWS Resilience Hub	resiliencehub
AWS Resource Explorer	resource-explorer-2
AWS Resource Groups	resource-groups

Service	Service prefix
AWS RoboMaker	robomaker
AWS Identity and Access Management Roles Anywhere	rolesanywhere
Amazon Route 53	route53
Amazon Route 53 Recovery Controls	route53-recovery-control-config
Amazon Route 53 Recovery Readiness	route53-recovery-readiness
Amazon Route 53 Resolver	route53resolver
AWS CloudWatch RUM	rum
Amazon Simple Storage Service	s3
Amazon S3 on Outposts	s3-outposts
Amazon SageMaker AI geospatial capabilities	sagemaker-geospatial
Savings Plans	savingsplans
Amazon EventBridge Schemas	schemas
Amazon SimpleDB	sdb
AWS Secrets Manager	secretsmanager
AWS Security Hub CSPM	securityhub
Amazon Security Lake	securitylake
AWS Serverless Application Repository	serverlessrepo
AWS Service Catalog	servicecatalog
AWS Cloud Map	servicediscovery

Service	Service prefix
Service Quotas	servicequotas
Amazon Simple Email Service	ses
AWS Shield	shield
AWS Signer	signer
AWS SimSpace Weaver	simspaceweaver
AWS Server Migration Service	sms
Amazon Pinpoint SMS and Voice Service	sms-voice
AWS Snowball Edge	snowball
Amazon Simple Queue Service	sqs
AWS Systems Manager	ssm
AWS Systems Manager Incident Manager	ssm-incidents
AWS Systems Manager for SAP	ssm-sap
AWS Step Functions	states
AWS Security Token Service	sts
Amazon Simple Workflow Service	swf
Amazon CloudWatch Synthetics	synthetics
AWS Resource Groups Tagging API	tag
Amazon Textract	textract
Amazon Timestream	timestream
AWS Telco Network Builder	tnb

Service	Service prefix
Amazon Transcribe	transcribe
AWS Transfer Family	transfer
Amazon Translate	translate
Connect Customer Voice ID	voiceid
Amazon VPC Lattice	vpc-lattice
AWS WAFV2	wafv2
AWS Well-Architected Tool	wellarchitected
Amazon Connect Customer Wisdom	wisdom
Amazon WorkLink	worklink
Amazon WorkSpaces	workspaces
AWS X-Ray	xray

Actions for action last accessed information

The following table lists the actions for which action last accessed information is available.

Important

The `iam:UpdateAccountName` action will be deprecated on April 22, 2026. After April 22, 2026, only the `account:PutAccountName` permission will control account name update access. We strongly recommend that you update any service control policies (SCPs) that control account name updates to use the `account:PutAccountName` permission.

Service prefix	Actions
access-analyzer	access-analyzer:ApplyArchiveRule

Service prefix	Actions
	access-analyzer:CancelPolicyGeneration
	access-analyzer:CheckAccessNotGranted
	access-analyzer:CheckNoNewAccess
	access-analyzer:CheckNoPublicAccess
	access-analyzer:CreateAccessPreview
	access-analyzer:CreateAnalyzer
	access-analyzer:CreateArchiveRule
	access-analyzer:DeleteAnalyzer
	access-analyzer:DeleteArchiveRule
	access-analyzer:GenerateFindingRecommendation
	access-analyzer:GetAccessPreview
	access-analyzer:GetAnalyzedResource
	access-analyzer:GetAnalyzer
	access-analyzer:GetArchiveRule
	access-analyzer:GetFinding
	access-analyzer:GetFindingRecommendation
	access-analyzer:GetFindingsStatistics
	access-analyzer:GetGeneratedPolicy
	access-analyzer:ListAccessPreviewFindings
	access-analyzer:ListAccessPreviews
	access-analyzer:ListAnalyzedResources

Service prefix	Actions
	access-analyzer:ListAnalyzers access-analyzer:ListArchiveRules access-analyzer:ListFindings access-analyzer:ListPolicyGenerations access-analyzer:StartPolicyGeneration access-analyzer:StartResourceScan access-analyzer:UpdateAnalyzer access-analyzer:UpdateArchiveRule access-analyzer:UpdateFindings access-analyzer:ValidatePolicy

Service prefix	Actions
account	account:AcceptPrimaryEmailUpdate account:DeleteAlternateContact account:DisableRegion account:EnableRegion account:GetAccountInformation account:GetAlternateContact account:GetContactInformation account:GetGovCloudAccountInformation account:GetPrimaryEmail account:GetRegionOptStatus account:ListRegions account:PutAccountName account:PutAlternateContact account:PutContactInformation account:StartPrimaryEmailUpdate

Service prefix	Actions
acm	acm:CreateAcmeEndpoint
	acm:DeleteCertificate
	acm:DescribeCertificate
	acm:ExportCertificate
	acm:GetAccountConfiguration
	acm:GetCertificate
	acm:ImportCertificate
	acm:ListAcmeEndpoints
	acm:ListCertificates
	acm:PutAccountConfiguration
	acm:RenewCertificate
	acm:RequestCertificate
	acm:ResendValidationEmail
	acm:SearchCertificates
	acm:UpdateCertificateOptions

Service prefix	Actions
airflow	airflow:CreateCliToken airflow:CreateEnvironment airflow:CreateWebLoginToken airflow:DeleteEnvironment airflow:GetEnvironment airflow:ListEnvironments airflow:PublishMetrics airflow:UpdateEnvironment

Service prefix	Actions
amplify	amplify:CreateApp
	amplify:CreateBackendEnvironment
	amplify:CreateBranch
	amplify:CreateDeployment
	amplify:CreateDomainAssociation
	amplify:CreateWebHook
	amplify:DeleteApp
	amplify:DeleteBackendEnvironment
	amplify:DeleteBranch
	amplify:DeleteDomainAssociation
	amplify:DeleteJob
	amplify:DeleteWebHook
	amplify:GenerateAccessLogs
	amplify:GetApp
	amplify:GetArtifactUrl
	amplify:GetBackendEnvironment
	amplify:GetBranch
	amplify:GetDomainAssociation
	amplify:GetJob
	amplify:GetWebHook
	amplify:ListApps

Service prefix	Actions
	amplify:ListArtifacts amplify:ListBackendEnvironments amplify:ListBranches amplify:ListDomainAssociations amplify:ListJobs amplify:ListWebHooks amplify:StartDeployment amplify:StartJob amplify:StopJob amplify:UpdateApp amplify:UpdateBranch amplify:UpdateDomainAssociation amplify:UpdateWebHook

Service prefix	Actions
amplifyuibuilder	amplifyuibuilder:CreateComponent amplifyuibuilder:CreateForm amplifyuibuilder:CreateTheme amplifyuibuilder:DeleteComponent amplifyuibuilder:DeleteForm amplifyuibuilder:DeleteTheme amplifyuibuilder:ExportComponents amplifyuibuilder:ExportThemes amplifyuibuilder:GetCodegenJob amplifyuibuilder:ListCodegenJobs amplifyuibuilder:ListComponents amplifyuibuilder:ListForms amplifyuibuilder:ListThemes amplifyuibuilder:ResetMetadataFlag amplifyuibuilder:StartCodegenJob amplifyuibuilder:UpdateComponent amplifyuibuilder:UpdateForm amplifyuibuilder:UpdateTheme

Service prefix	Actions
app-integrations	app-integrations:CreateApplication
	app-integrations:CreateDataIntegration
	app-integrations:CreateDataIntegrationAssociation
	app-integrations:CreateEventIntegration
	app-integrations:DeleteApplication
	app-integrations:DeleteDataIntegration
	app-integrations:DeleteEventIntegration
	app-integrations:GetApplication
	app-integrations:GetDataIntegration
	app-integrations:GetEventIntegration
	app-integrations:ListApplicationAssociations
	app-integrations:ListApplications
	app-integrations:ListDataIntegrationAssociations
	app-integrations:ListDataIntegrations
	app-integrations:ListEventIntegrationAssociations
	app-integrations:ListEventIntegrations
	app-integrations:UpdateApplication
	app-integrations:UpdateDataIntegration
	app-integrations:UpdateDataIntegrationAssociation
	app-integrations:UpdateEventIntegration

Service prefix	Actions
appconfig	appconfig:CreateApplication appconfig:CreateConfigurationProfile appconfig:CreateDeploymentStrategy appconfig:CreateEnvironment appconfig:CreateExtension appconfig:CreateExtensionAssociation appconfig:CreateHostedConfigurationVersion appconfig:DeleteApplication appconfig:DeleteConfigurationProfile appconfig:DeleteDeploymentStrategy appconfig:DeleteEnvironment appconfig:DeleteExtension appconfig:DeleteExtensionAssociation appconfig:DeleteHostedConfigurationVersion appconfig:GetAccountSettings appconfig:GetApplication appconfig:GetConfiguration appconfig:GetConfigurationProfile appconfig:GetDeployment appconfig:GetDeploymentStrategy appconfig:GetEnvironment

Service prefix	Actions
	appconfig:GetExtension appconfig:GetExtensionAssociation appconfig:GetHostedConfigurationVersion appconfig:ListApplications appconfig:ListConfigurationProfiles appconfig:ListDeploymentStrategies appconfig:ListDeployments appconfig:ListEnvironments appconfig:ListExtensionAssociations appconfig:ListExtensions appconfig:ListHostedConfigurationVersions appconfig:StartDeployment appconfig:StopDeployment appconfig:UpdateAccountSettings appconfig:UpdateApplication appconfig:UpdateConfigurationProfile appconfig:UpdateDeploymentStrategy appconfig:UpdateEnvironment appconfig:UpdateExtension appconfig:UpdateExtensionAssociation appconfig:ValidateConfiguration

Service prefix	Actions
appflow	appflow:CancelFlowExecutions
	appflow:CreateConnectorProfile
	appflow:CreateFlow
	appflow:DeleteConnectorProfile
	appflow:DeleteFlow
	appflow:DescribeConnector
	appflow:DescribeConnectorEntity
	appflow:DescribeConnectorProfiles
	appflow:DescribeConnectors
	appflow:DescribeFlow
	appflow:DescribeFlowExecutionRecords
	appflow:ListConnectorEntities
	appflow:ListConnectors
	appflow:ListFlows
	appflow:RegisterConnector
	appflow:ResetConnectorMetadataCache
	appflow:StartFlow
	appflow:StopFlow
	appflow:UnRegisterConnector
	appflow:UpdateConnectorProfile
	appflow:UpdateConnectorRegistration

Service prefix	Actions
	appflow:UpdateFlow

Service prefix	Actions
applicationinsights	applicationinsights:AddWorkload applicationinsights:CreateApplication applicationinsights:CreateComponent applicationinsights:CreateLogPattern applicationinsights:DeleteApplication applicationinsights:DeleteComponent applicationinsights:DeleteLogPattern applicationinsights:DescribeApplication applicationinsights:DescribeComponent applicationinsights:DescribeComponentConfiguration applicationinsights:DescribeComponentConfigurationRecommendation applicationinsights:DescribeLogPattern applicationinsights:DescribeObservation applicationinsights:DescribeProblem applicationinsights:DescribeProblemObservations applicationinsights:DescribeWorkload applicationinsights:ListApplications applicationinsights:ListComponents applicationinsights:ListConfigurationHistory applicationinsights:ListLogPatternSets

Service prefix	Actions
	applicationinsights:ListLogPatterns applicationinsights:ListProblems applicationinsights:ListWorkloads applicationinsights:RemoveWorkload applicationinsights:UpdateApplication applicationinsights:UpdateComponent applicationinsights:UpdateComponentConfiguration applicationinsights:UpdateLogPattern applicationinsights:UpdateWorkload

Service prefix	Actions
appmesh	appmesh:CreateGatewayRoute
	appmesh:CreateMesh
	appmesh:CreateRoute
	appmesh:CreateVirtualGateway
	appmesh:CreateVirtualNode
	appmesh:CreateVirtualRouter
	appmesh:CreateVirtualService
	appmesh:DeleteGatewayRoute
	appmesh:DeleteMesh
	appmesh:DeleteRoute
	appmesh:DeleteVirtualGateway
	appmesh:DeleteVirtualNode
	appmesh:DeleteVirtualRouter
	appmesh:DeleteVirtualService
	appmesh:DescribeGatewayRoute
	appmesh:DescribeMesh
	appmesh:DescribeRoute
	appmesh:DescribeVirtualGateway
	appmesh:DescribeVirtualNode
	appmesh:DescribeVirtualRouter
	appmesh:DescribeVirtualService

Service prefix	Actions
	appmesh:ListGatewayRoutes
	appmesh:ListMeshes
	appmesh:ListRoutes
	appmesh:ListVirtualGateways
	appmesh:ListVirtualNodes
	appmesh:ListVirtualRouters
	appmesh:ListVirtualServices
	appmesh:StreamAggregatedResources
	appmesh:UpdateGatewayRoute
	appmesh:UpdateMesh
	appmesh:UpdateRoute
	appmesh:UpdateVirtualGateway
	appmesh:UpdateVirtualNode
	appmesh:UpdateVirtualRouter
	appmesh:UpdateVirtualService

Service prefix	Actions
appstream	appstream:AssociateAppBlockBuilderAppBlock appstream:AssociateApplicationFleet appstream:AssociateApplicationToEntitlement appstream:AssociateFleet appstream:AssociateSoftwareToImageBuilder appstream:BatchAssociateUserStack appstream:BatchDisassociateUserStack appstream:CopyImage appstream:CreateAppBlock appstream:CreateAppBlockBuilder appstream:CreateAppBlockBuilderStreamingURL appstream:CreateApplication appstream:CreateDirectoryConfig appstream:CreateEntitlement appstream:CreateFleet appstream:CreateImageBuilder appstream:CreateImageBuilderStreamingURL appstream:CreateStack appstream:CreateStreamingURL appstream:CreateThemeForStack appstream:CreateUpdatedImage

Service prefix	Actions
	appstream:CreateUsageReportSubscription appstream:CreateUser appstream:DeleteAppBlock appstream:DeleteAppBlockBuilder appstream:DeleteApplication appstream:DeleteDirectoryConfig appstream:DeleteEntitlement appstream:DeleteFleet appstream:DeleteImage appstream:DeleteImageBuilder appstream:DeleteImagePermissions appstream:DeleteStack appstream:DeleteThemeForStack appstream:DeleteUsageReportSubscription appstream:DeleteUser appstream:DescribeAppBlockBuilderAppBlockAssociations appstream:DescribeAppBlockBuilders appstream:DescribeAppBlocks appstream:DescribeAppLicenseUsage appstream:DescribeApplicationFleetAssociations appstream:DescribeApplications

Service prefix	Actions
	appstream:DescribeDirectoryConfigs appstream:DescribeEntitlements appstream:DescribeFleets appstream:DescribeImageBuilders appstream:DescribeImagePermissions appstream:DescribeImages appstream:DescribeSessions appstream:DescribeStacks appstream:DescribeThemeForStack appstream:DescribeUsageReportSubscriptions appstream:DescribeUserStackAssociations appstream:DescribeUsers appstream:DisableUser appstream:DisassociateAppBlockBuilderAppBlock appstream:DisassociateApplicationFleet appstream:DisassociateApplicationFromEntitlement appstream:DisassociateFleet appstream:DisassociateSoftwareFromImageBuilder appstream:EnableUser appstream:ExpireSession appstream:GetExportImageTask

Service prefix	Actions
	appstream:ListAssociatedFleets appstream:ListAssociatedStacks appstream:ListEntitledApplications appstream:ListExportImageTasks appstream:StartAppBlockBuilder appstream:StartFleet appstream:StartImageBuilder appstream:StartSoftwareDeploymentToImageBuilder appstream:StopAppBlockBuilder appstream:StopFleet appstream:StopImageBuilder appstream:UpdateAppBlockBuilder appstream:UpdateApplication appstream:UpdateDirectoryConfig appstream:UpdateEntitlement appstream:UpdateFleet appstream:UpdateImagePermissions appstream:UpdateStack appstream:UpdateThemeForStack

Service prefix	Actions
appsync	appsync:AssociateApi
	appsync:AssociateMergedGraphqlApi
	appsync:AssociateSourceGraphqlApi
	appsync:CreateApi
	appsync:CreateApiCache
	appsync:CreateApiKey
	appsync:CreateChannelNamespace
	appsync:CreateDataSource
	appsync:CreateDomainName
	appsync:CreateFunction
	appsync:CreateGraphqlApi
	appsync:CreateResolver
	appsync:CreateType
	appsync:DeleteApi
	appsync:DeleteApiCache
	appsync:DeleteApiKey
	appsync:DeleteChannelNamespace
	appsync:DeleteDataSource
	appsync:DeleteDomainName
	appsync:DeleteFunction
	appsync:DeleteGraphqlApi

Service prefix	Actions
	appsync:DeleteResolver
	appsync:DeleteType
	appsync:DisassociateApi
	appsync:DisassociateMergedGraphqlApi
	appsync:DisassociateSourceGraphqlApi
	appsync:EvaluateCode
	appsync:EvaluateMappingTemplate
	appsync:FlushApiCache
	appsync:GetApi
	appsync:GetApiAssociation
	appsync:GetApiCache
	appsync:GetChannelNamespace
	appsync:GetDataSource
	appsync:GetDataSourceIntrospection
	appsync:GetDomainName
	appsync:GetFunction
	appsync:GetGraphqlApi
	appsync:GetGraphqlApiEnvironmentVariables
	appsync:GetIntrospectionSchema
	appsync:GetResolver
	appsync:GetSchemaCreationStatus

Service prefix	Actions
	appsync:GetSourceApiAssociation
	appsync:GetType
	appsync:ListApiKeys
	appsync:ListApis
	appsync:ListChannelNamespaces
	appsync:ListDataSources
	appsync:ListDomainNames
	appsync:ListFunctions
	appsync:ListGraphqlApis
	appsync:ListResolvers
	appsync:ListResolversByFunction
	appsync:ListSourceApiAssociations
	appsync:ListTypes
	appsync:ListTypesByAssociation
	appsync:PutGraphqlApiEnvironmentVariables
	appsync:StartDataSourceIntrospection
	appsync:StartSchemaCreation
	appsync:StartSchemaMerge
	appsync:UpdateApi
	appsync:UpdateApiCache
	appsync:UpdateApiKey

Service prefix	Actions
	appsync:UpdateChannelNamespace appsync:UpdateDataSource appsync:UpdateDomainName appsync:UpdateFunction appsync:UpdateGraphqlApi appsync:UpdateResolver appsync:UpdateSourceApiAssociation appsync:UpdateType

Service prefix	Actions
aps	aps:CreateAlertManagerDefinition aps:CreateAnomalyDetector aps:CreateLoggingConfiguration aps:CreateQueryLoggingConfiguration aps:CreateRuleGroupsNamespace aps:CreateWorkspace aps:DeleteAlertManagerDefinition aps:DeleteAnomalyDetector aps:DeleteLoggingConfiguration aps:DeleteQueryLoggingConfiguration aps:DeleteResourcePolicy aps:DeleteRuleGroupsNamespace aps:DeleteScraper aps:DeleteScraperLoggingConfiguration aps:DeleteWorkspace aps:DescribeAlertManagerDefinition aps:DescribeAnomalyDetector aps:DescribeLoggingConfiguration aps:DescribeQueryLoggingConfiguration aps:DescribeResourcePolicy aps:DescribeRuleGroupsNamespace

Service prefix	Actions
	aps:DescribeScraper aps:DescribeScraperLoggingConfiguration aps:DescribeWorkspace aps:DescribeWorkspaceConfiguration aps:GetDefaultScraperConfiguration aps:ListAnomalyDetectors aps:ListRuleGroupsNamespaces aps:ListScrapers aps:ListWorkspaces aps:PutAlertManagerDefinition aps:PutAnomalyDetector aps:PutResourcePolicy aps:PutRuleGroupsNamespace aps:UpdateLoggingConfiguration aps:UpdateQueryLoggingConfiguration aps:UpdateScraper aps:UpdateScraperLoggingConfiguration aps:UpdateWorkspaceAlias aps:UpdateWorkspaceConfiguration

Service prefix	Actions
athena	athena:BatchGetNamedQuery
	athena:BatchGetPreparedStatement
	athena:BatchGetQueryExecution
	athena:CancelCapacityReservation
	athena:CreateCapacityReservation
	athena:CreateDataCatalog
	athena:CreateNamedQuery
	athena:CreateNotebook
	athena:CreatePreparedStatement
	athena:CreatePresignedNotebookUrl
	athena:CreateWorkGroup
	athena:DeleteCapacityReservation
	athena:DeleteDataCatalog
	athena:DeleteNamedQuery
	athena:DeleteNotebook
	athena:DeletePreparedStatement
	athena:DeleteWorkGroup
	athena:ExportNotebook
	athena:GetCalculationExecution
	athena:GetCalculationExecutionCode
	athena:GetCalculationExecutionStatus

Service prefix	Actions
	athena:GetCapacityAssignmentConfiguration
	athena:GetCapacityReservation
	athena:GetDataCatalog
	athena:GetDatabase
	athena:GetNamedQuery
	athena:GetNotebookMetadata
	athena:GetPreparedStatement
	athena:GetQueryExecution
	athena:GetQueryResults
	athena:GetQueryResultsStream
	athena:GetQueryRuntimeStatistics
	athena:GetResourceDashboard
	athena:GetSession
	athena:GetSessionEndpoint
	athena:GetSessionStatus
	athena:GetTableMetadata
	athena:GetWorkGroup
	athena:ImportNotebook
	athena:ListApplicationDPUSizes
	athena:ListCalculationExecutions
	athena:ListCapacityReservations

Service prefix	Actions
	athena:ListDataCatalogs
	athena:ListDatabases
	athena:ListEngineVersions
	athena:ListExecutors
	athena:ListNamedQueries
	athena:ListNotebookMetadata
	athena:ListNotebookSessions
	athena:ListPreparedStatements
	athena:ListQueryExecutions
	athena:ListSessions
	athena:ListTableMetadata
	athena:ListWorkGroups
	athena:PutCapacityAssignmentConfiguration
	athena:StartCalculationExecution
	athena:StartQueryExecution
	athena:StartSession
	athena:StopCalculationExecution
	athena:StopQueryExecution
	athena:TerminateSession
	athena:UpdateCapacityReservation
	athena:UpdateDataCatalog

Service prefix	Actions
	athena:UpdateNamedQuery athena:UpdateNotebook athena:UpdateNotebookMetadata athena:UpdatePreparedStatement athena:UpdateWorkGroup

Service prefix	Actions
auditmanager	auditmanager:AssociateAssessmentReportEvidenceFolder auditmanager:BatchAssociateAssessmentReportEvidence auditmanager:BatchCreateDelegationByAssessment auditmanager:BatchDeleteDelegationByAssessment auditmanager:BatchDisassociateAssessmentReportEvidence auditmanager:BatchImportEvidenceToAssessmentControl auditmanager:CreateAssessment auditmanager:CreateAssessmentFramework auditmanager:CreateAssessmentReport auditmanager:CreateControl auditmanager:DeleteAssessment auditmanager:DeleteAssessmentFramework auditmanager:DeleteAssessmentFrameworkShare auditmanager:DeleteAssessmentReport auditmanager:DeleteControl auditmanager:DeregisterAccount auditmanager:DeregisterOrganizationAdminAccount auditmanager:DisassociateAssessmentReportEvidenceFolder auditmanager:GetAccountStatus auditmanager:GetAssessment auditmanager:GetAssessmentFramework

Service prefix	Actions
	auditmanager:GetAssessmentReportUrl auditmanager:GetChangeLogs auditmanager:GetControl auditmanager:GetDelegations auditmanager:GetEvidence auditmanager:GetEvidenceByEvidenceFolder auditmanager:GetEvidenceFileUploadUrl auditmanager:GetEvidenceFolder auditmanager:GetEvidenceFoldersByAssessment auditmanager:GetEvidenceFoldersByAssessmentControl auditmanager:GetInsights auditmanager:GetInsightsByAssessment auditmanager:GetOrganizationAdminAccount auditmanager:GetServicesInScope auditmanager:GetSettings auditmanager:ListAssessmentControlInsightsByControlDomain auditmanager:ListAssessmentFrameworkShareRequests auditmanager:ListAssessmentFrameworks auditmanager:ListAssessmentReports auditmanager:ListAssessments auditmanager:ListControlDomainInsights

Service prefix	Actions
	auditmanager:ListControlDomainInsightsByAssessment auditmanager:ListControlInsightsByControlDomain auditmanager:ListControls auditmanager:ListKeywordsForDataSource auditmanager:ListNotifications auditmanager:RegisterAccount auditmanager:RegisterOrganizationAdminAccount auditmanager:StartAssessmentFrameworkShare auditmanager:UpdateAssessment auditmanager:UpdateAssessmentControl auditmanager:UpdateAssessmentControlSetStatus auditmanager:UpdateAssessmentFramework auditmanager:UpdateAssessmentFrameworkShare auditmanager:UpdateAssessmentStatus auditmanager:UpdateControl auditmanager:UpdateSettings auditmanager:ValidateAssessmentReportIntegrity

Service prefix	Actions
autoscaling	autoscaling:AttachInstances
	autoscaling:AttachLoadBalancerTargetGroups
	autoscaling:AttachLoadBalancers
	autoscaling:AttachTrafficSources
	autoscaling:BatchDeleteScheduledAction
	autoscaling:BatchPutScheduledUpdateGroupAction
	autoscaling:CancelInstanceRefresh
	autoscaling:CompleteLifecycleAction
	autoscaling:CreateAutoScalingGroup
	autoscaling:CreateLaunchConfiguration
	autoscaling:DeleteAutoScalingGroup
	autoscaling:DeleteLaunchConfiguration
	autoscaling:DeleteLifecycleHook
	autoscaling:DeleteNotificationConfiguration
	autoscaling:DeletePolicy
	autoscaling:DeleteScheduledAction
	autoscaling:DeleteWarmPool
	autoscaling:DescribeAccountLimits
	autoscaling:DescribeAdjustmentTypes
	autoscaling:DescribeAutoScalingGroups
	autoscaling:DescribeAutoScalingInstances

Service prefix	Actions
	autoscaling:DescribeAutoScalingNotificationTypes
	autoscaling:DescribeInstanceRefreshes
	autoscaling:DescribeLaunchConfigurations
	autoscaling:DescribeLifecycleHookTypes
	autoscaling:DescribeLifecycleHooks
	autoscaling:DescribeLoadBalancerTargetGroups
	autoscaling:DescribeLoadBalancers
	autoscaling:DescribeMetricCollectionTypes
	autoscaling:DescribeNotificationConfigurations
	autoscaling:DescribePolicies
	autoscaling:DescribeScalingActivities
	autoscaling:DescribeScalingProcessTypes
	autoscaling:DescribeScheduledActions
	autoscaling:DescribeTerminationPolicyTypes
	autoscaling:DescribeTrafficSources
	autoscaling:DescribeWarmPool
	autoscaling:DetachInstances
	autoscaling:DetachLoadBalancerTargetGroups
	autoscaling:DetachLoadBalancers
	autoscaling:DetachTrafficSources
	autoscaling:DisableMetricsCollection

Service prefix	Actions
	autoscaling:EnableMetricsCollection autoscaling:EnterStandby autoscaling:ExecutePolicy autoscaling:ExitStandby autoscaling:GetPredictiveScalingForecast autoscaling:PutLifecycleHook autoscaling:PutNotificationConfiguration autoscaling:PutScalingPolicy autoscaling:PutScheduledUpdateGroupAction autoscaling:PutWarmPool autoscaling:RecordLifecycleActionHeartbeat autoscaling:ResumeProcesses autoscaling:RollbackInstanceRefresh autoscaling:SetDesiredCapacity autoscaling:SetInstanceHealth autoscaling:SetInstanceProtection autoscaling:StartInstanceRefresh autoscaling:SuspendProcesses autoscaling:TerminateInstanceInAutoScalingGroup autoscaling:UpdateAutoScalingGroup
aws-marketplace	aws-marketplace:GetEntitlements

Service prefix	Actions
backup	backup:CancelLegalHold
	backup:CreateBackupPlan
	backup:CreateBackupSelection
	backup:CreateBackupVault
	backup:CreateFramework
	backup:CreateLegalHold
	backup:CreateLogicallyAirGappedBackupVault
	backup:CreateReportPlan
	backup:CreateRestoreAccessBackupVault
	backup:CreateRestoreTestingPlan
	backup:CreateRestoreTestingSelection
	backup:CreateTieringConfiguration
	backup:DeleteBackupPlan
	backup:DeleteBackupSelection
	backup:DeleteBackupVault
	backup:DeleteBackupVaultAccessPolicy
	backup:DeleteBackupVaultLockConfiguration
	backup:DeleteBackupVaultNotifications
	backup:DeleteFramework
	backup:DeleteRecoveryPoint
	backup:DeleteReportPlan

Service prefix	Actions
	backup:DeleteRestoreTestingPlan
	backup:DeleteRestoreTestingSelection
	backup:DeleteTieringConfiguration
	backup:DescribeBackupJob
	backup:DescribeBackupVault
	backup:DescribeCopyJob
	backup:DescribeFramework
	backup:DescribeGlobalSettings
	backup:DescribeProtectedResource
	backup:DescribeRecoveryPoint
	backup:DescribeRegionSettings
	backup:DescribeReportJob
	backup:DescribeReportPlan
	backup:DescribeRestoreJob
	backup:DescribeScanJob
	backup:DisassociateRecoveryPoint
	backup:DisassociateRecoveryPointFromParent
	backup:ExportBackupPlanTemplate
	backup:GetBackupPlan
	backup:GetBackupPlanFromJSON
	backup:GetBackupPlanFromTemplate

Service prefix	Actions
	backup:GetBackupSelection
	backup:GetBackupVaultAccessPolicy
	backup:GetBackupVaultNotifications
	backup:GetLegalHold
	backup:GetRecoveryPointRestoreMetadata
	backup:GetRestoreJobMetadata
	backup:GetRestoreTestingInferredMetadata
	backup:GetRestoreTestingPlan
	backup:GetRestoreTestingSelection
	backup:GetSupportedResourceTypes
	backup:GetTieringConfiguration
	backup:ListBackupJobSummaries
	backup:ListBackupJobs
	backup:ListBackupPlanTemplates
	backup:ListBackupPlanVersions
	backup:ListBackupPlans
	backup:ListBackupSelections
	backup:ListBackupVaults
	backup:ListCopyJobSummaries
	backup:ListCopyJobs
	backup:ListFrameworks

Service prefix	Actions
	backup:ListIndexedRecoveryPoints
	backup:ListLegalHolds
	backup:ListProtectedResources
	backup:ListRecoveryPointsByBackupVault
	backup:ListRecoveryPointsByLegalHold
	backup:ListRecoveryPointsByResource
	backup:ListReportJobs
	backup:ListReportPlans
	backup:ListRestoreAccessBackupVaults
	backup:ListRestoreJobSummaries
	backup:ListRestoreJobs
	backup:ListRestoreJobsByProtectedResource
	backup:ListRestoreTestingPlans
	backup:ListRestoreTestingSelections
	backup:ListScanJobSummaries
	backup:ListScanJobs
	backup:ListTieringConfigurations
	backup:PutBackupVaultAccessPolicy
	backup:PutBackupVaultLockConfiguration
	backup:PutBackupVaultNotifications
	backup:PutRestoreValidationResult

Service prefix	Actions
	backup:StartBackupJob
	backup:StartCopyJob
	backup:StartReportJob
	backup:StartRestoreJob
	backup:StopBackupJob
	backup:UpdateBackupPlan
	backup:UpdateFramework
	backup:UpdateGlobalSettings
	backup:UpdateRecoveryPointLifecycle
	backup:UpdateRegionSettings
	backup:UpdateReportPlan
	backup:UpdateRestoreTestingPlan
	backup:UpdateRestoreTestingSelection
	backup:UpdateTieringConfiguration

Service prefix	Actions
batch	batch:CancelJob batch:CreateComputeEnvironment batch:CreateConsumableResource batch:CreateJobQueue batch:CreateQuotaShare batch:CreateSchedulingPolicy batch:CreateServiceEnvironment batch:DeleteComputeEnvironment batch:DeleteConsumableResource batch:DeleteJobQueue batch:DeleteQuotaShare batch:DeleteSchedulingPolicy batch:DeleteServiceEnvironment batch:DeregisterJobDefinition batch:DescribeComputeEnvironments batch:DescribeConsumableResource batch:DescribeJobDefinitions batch:DescribeJobQueues batch:DescribeJobs batch:DescribeQuotaShare batch:DescribeSchedulingPolicies

Service prefix	Actions
	batch:DescribeServiceEnvironments batch:DescribeServiceJob batch:GetJobQueueSnapshot batch:ListConsumableResources batch:ListJobs batch:ListJobsByConsumableResource batch:ListQuotaShares batch:ListSchedulingPolicies batch:ListServiceJobs batch:RegisterJobDefinition batch:SubmitJob batch:SubmitServiceJob batch:TerminateJob batch:TerminateServiceJob batch:UpdateComputeEnvironment batch:UpdateConsumableResource batch:UpdateJobQueue batch:UpdateQuotaShare batch:UpdateSchedulingPolicy batch:UpdateServiceEnvironment batch:UpdateServiceJob

Service prefix	Actions
braket	braket:CancelJob
	braket:CancelQuantumTask
	braket:CreateJob
	braket:CreateQuantumTask
	braket:CreateSpendingLimit
	braket:GetDevice
	braket:GetJob
	braket:GetQuantumTask
	braket:SearchDevices
	braket:SearchJobs
	braket:SearchQuantumTasks
	braket:SearchSpendingLimits

Service prefix	Actions
budgets	budgets:CreateBudgetAction budgets:DeleteBudgetAction budgets:DescribeBudgetAction budgets:DescribeBudgetActionHistories budgets:DescribeBudgetActionsForAccount budgets:DescribeBudgetActionsForBudget budgets:ExecuteBudgetAction budgets:ModifyBudget budgets:UpdateBudgetAction budgets:ViewBudget
cloud9	cloud9:CreateEnvironmentEC2 cloud9:CreateEnvironmentMembership cloud9:DeleteEnvironment cloud9:DeleteEnvironmentMembership cloud9:DescribeEnvironmentMemberships cloud9:DescribeEnvironmentStatus cloud9:DescribeEnvironments cloud9:ListEnvironments cloud9:UpdateEnvironment cloud9:UpdateEnvironmentMembership

Service prefix	Actions
cloudformation	cloudformation:BatchDescribeTypeConfigurations cloudformation:CancelUpdateStack cloudformation:ContinueUpdateRollback cloudformation:CreateChangeSet cloudformation:CreateGeneratedTemplate cloudformation:CreateStack cloudformation:CreateStackInstances cloudformation:CreateStackSet cloudformation:DeactivateType cloudformation:DeleteChangeSet cloudformation:DeleteGeneratedTemplate cloudformation:DeleteStack cloudformation:DeleteStackInstances cloudformation:DeleteStackSet cloudformation:DeregisterType cloudformation:DescribeAccountLimits cloudformation:DescribeChangeSet cloudformation:DescribeChangeSetHooks cloudformation:DescribeEvents cloudformation:DescribeGeneratedTemplate cloudformation:DescribeOrganizationsAccess

Service prefix	Actions
	cloudformation:DescribePublisher cloudformation:DescribeResourceScan cloudformation:DescribeStackDriftDetectionStatus cloudformation:DescribeStackEvents cloudformation:DescribeStackInstance cloudformation:DescribeStackResource cloudformation:DescribeStackResourceDrifts cloudformation:DescribeStackResources cloudformation:DescribeStackSet cloudformation:DescribeStackSetOperation cloudformation:DescribeStacks cloudformation:DescribeType cloudformation:DescribeTypeRegistration cloudformation:DetectStackDrift cloudformation:DetectStackResourceDrift cloudformation:DetectStackSetDrift cloudformation:EstimateTemplateCost cloudformation:ExecuteChangeSet cloudformation:GetGeneratedTemplate cloudformation:GetHookResult cloudformation:GetStackPolicy

Service prefix	Actions
	cloudformation:GetTemplate cloudformation:GetTemplateSummary cloudformation:ImportStacksToStackSet cloudformation:ListChangeSets cloudformation:ListExports cloudformation:ListGeneratedTemplates cloudformation:ListHookResults cloudformation:ListImports cloudformation:ListResourceScanRelatedResources cloudformation:ListResourceScanResources cloudformation:ListResourceScans cloudformation:ListStackInstanceResourceDrifts cloudformation:ListStackInstances cloudformation:ListStackRefactors cloudformation:ListStackResources cloudformation:ListStackSetAutoDeploymentTargets cloudformation:ListStackSetOperationResults cloudformation:ListStackSetOperations cloudformation:ListStackSets cloudformation:ListTypeRegistrations cloudformation:ListTypeVersions

Service prefix	Actions
	cloudformation:ListTypes cloudformation:PublishType cloudformation:RecordHandlerProgress cloudformation:RegisterPublisher cloudformation:RegisterType cloudformation:RollbackStack cloudformation:SetStackPolicy cloudformation:SetTypeConfiguration cloudformation:SetTypeDefaultVersion cloudformation:SignalResource cloudformation:StartResourceScan cloudformation:StopStackSetOperation cloudformation:TestType cloudformation:UpdateGeneratedTemplate cloudformation:UpdateStack cloudformation:UpdateStackInstances cloudformation:UpdateStackSet cloudformation:UpdateTerminationProtection cloudformation:ValidateTemplate

Service prefix	Actions
cloudfront	cloudfront:AssociateAlias
	cloudfront:AssociateDistributionTenantWebACL
	cloudfront:AssociateDistributionWebACL
	cloudfront:CreateCachePolicy
	cloudfront:CreateCloudFrontOriginAccessIdentity
	cloudfront:CreateConnectionFunction
	cloudfront:CreateContinuousDeploymentPolicy
	cloudfront:CreateDistributionTenant
	cloudfront:CreateFieldLevelEncryptionConfig
	cloudfront:CreateFieldLevelEncryptionProfile
	cloudfront:CreateFunction
	cloudfront:CreateInvalidation
	cloudfront:CreateKeyGroup
	cloudfront:CreateKeyValueStore
	cloudfront:CreateMonitoringSubscription
	cloudfront:CreateOriginAccessControl
	cloudfront:CreateOriginRequestPolicy
	cloudfront:CreatePublicKey
	cloudfront:CreateRealtimeLogConfig
	cloudfront:CreateResponseHeadersPolicy
	cloudfront:CreateTrustStore

Service prefix	Actions
	cloudfront:DeleteAnycastIpList
	cloudfront:DeleteCachePolicy
	cloudfront:DeleteCloudFrontOriginAccessIdentity
	cloudfront:DeleteConnectionFunction
	cloudfront:DeleteConnectionGroup
	cloudfront:DeleteContinuousDeploymentPolicy
	cloudfront:DeleteDistribution
	cloudfront:DeleteDistributionTenant
	cloudfront:DeleteFieldLevelEncryptionConfig
	cloudfront:DeleteFieldLevelEncryptionProfile
	cloudfront:DeleteFunction
	cloudfront:DeleteKeyGroup
	cloudfront:DeleteKeyValueStore
	cloudfront:DeleteMonitoringSubscription
	cloudfront:DeleteOriginAccessControl
	cloudfront:DeleteOriginRequestPolicy
	cloudfront:DeletePublicKey
	cloudfront:DeleteRealtimeLogConfig
	cloudfront:DeleteResponseHeadersPolicy
	cloudfront:DeleteStreamingDistribution
	cloudfront:DeleteTrustStore

Service prefix	Actions
	cloudfront:DeleteVpcOrigin
	cloudfront:DescribeFunction
	cloudfront:DescribeKeyValueStore
	cloudfront:DisassociateDistributionTenantWebACL
	cloudfront:DisassociateDistributionWebACL
	cloudfront:GetAnycastIpList
	cloudfront:GetCachePolicy
	cloudfront:GetCachePolicyConfig
	cloudfront:GetCloudFrontOriginAccessIdentity
	cloudfront:GetCloudFrontOriginAccessIdentityConfig
	cloudfront:GetContinuousDeploymentPolicy
	cloudfront:GetContinuousDeploymentPolicyConfig
	cloudfront:GetDistributionConfig
	cloudfront:GetFieldLevelEncryption
	cloudfront:GetFieldLevelEncryptionConfig
	cloudfront:GetFieldLevelEncryptionProfile
	cloudfront:GetFieldLevelEncryptionProfileConfig
	cloudfront:GetFunction
	cloudfront:GetInvalidation
	cloudfront:GetInvalidationForDistributionTenant
	cloudfront:GetKeyGroup

Service prefix	Actions
	cloudfront:GetKeyGroupConfig
	cloudfront:GetMonitoringSubscription
	cloudfront:GetOriginAccessControl
	cloudfront:GetOriginAccessControlConfig
	cloudfront:GetOriginRequestPolicy
	cloudfront:GetOriginRequestPolicyConfig
	cloudfront:GetPublicKey
	cloudfront:GetPublicKeyConfig
	cloudfront:GetRealtimeLogConfig
	cloudfront:GetResponseHeadersPolicy
	cloudfront:GetResponseHeadersPolicyConfig
	cloudfront:GetStreamingDistribution
	cloudfront:GetStreamingDistributionConfig
	cloudfront:GetVpcOrigin
	cloudfront:ListAnycastIpLists
	cloudfront:ListCachePolicies
	cloudfront:ListCloudFrontOriginAccessIdentities
	cloudfront:ListConflictingAliases
	cloudfront:ListConnectionFunctions
	cloudfront:ListConnectionGroups
	cloudfront:ListContinuousDeploymentPolicies

Service prefix	Actions
	cloudfront:ListDistributionTenants
	cloudfront:ListDistributionTenantsByCustomization
	cloudfront:ListDistributions
	cloudfront:ListDistributionsByAnycastIpListId
	cloudfront:ListDistributionsByCachePolicyId
	cloudfront:ListDistributionsByConnectionMode
	cloudfront:ListDistributionsByKeyGroup
	cloudfront:ListDistributionsByOriginRequestPolicyId
	cloudfront:ListDistributionsByRealtimeLogConfig
	cloudfront:ListDistributionsByResponseHeadersPolicyId
	cloudfront:ListDistributionsByVpcOriginId
	cloudfront:ListDistributionsByWebACLId
	cloudfront:ListFieldLevelEncryptionConfigs
	cloudfront:ListFieldLevelEncryptionProfiles
	cloudfront:ListFunctions
	cloudfront:ListInvalidations
	cloudfront:ListInvalidationsForDistributionTenant
	cloudfront:ListKeyGroups
	cloudfront:ListKeyValueStores
	cloudfront:ListOriginAccessControls
	cloudfront:ListOriginRequestPolicies

Service prefix	Actions
	cloudfront:ListPublicKeys
	cloudfront:ListRealtimeLogConfigs
	cloudfront:ListResponseHeadersPolicies
	cloudfront:ListStreamingDistributions
	cloudfront:ListTrustStores
	cloudfront:PublishConnectionFunction
	cloudfront:PublishFunction
	cloudfront:TestConnectionFunction
	cloudfront:TestFunction
	cloudfront:UpdateAnycastIpList
	cloudfront:UpdateCachePolicy
	cloudfront:UpdateCloudFrontOriginAccessIdentity
	cloudfront:UpdateConnectionFunction
	cloudfront:UpdateConnectionGroup
	cloudfront:UpdateContinuousDeploymentPolicy
	cloudfront:UpdateDistribution
	cloudfront:UpdateDistributionTenant
	cloudfront:UpdateFieldLevelEncryptionConfig
	cloudfront:UpdateFieldLevelEncryptionProfile
	cloudfront:UpdateFunction
	cloudfront:UpdateKeyGroup

Service prefix	Actions
	cloudfront:UpdateKeyValueStore cloudfront:UpdateOriginAccessControl cloudfront:UpdateOriginRequestPolicy cloudfront:UpdatePublicKey cloudfront:UpdateRealtimeLogConfig cloudfront:UpdateResponseHeadersPolicy cloudfront:UpdateTrustStore
cloudhsm	cloudhsm:CreateHsm cloudhsm:DeleteBackup cloudhsm:DeleteHsm cloudhsm:DeleteResourcePolicy cloudhsm:DescribeBackups cloudhsm:DescribeClusters cloudhsm:GetResourcePolicy cloudhsm:InitializeCluster cloudhsm:ModifyBackupAttributes cloudhsm:ModifyCluster cloudhsm:PutResourcePolicy cloudhsm:RestoreBackup

Service prefix	Actions
cloudsearch	cloudsearch:BuildSuggesters
	cloudsearch:CreateDomain
	cloudsearch:DefineAnalysisScheme
	cloudsearch:DefineExpression
	cloudsearch:DefineIndexField
	cloudsearch:DefineSuggester
	cloudsearch:DeleteAnalysisScheme
	cloudsearch:DeleteDomain
	cloudsearch:DeleteExpression
	cloudsearch:DeleteIndexField
	cloudsearch:DeleteSuggester
	cloudsearch:DescribeAnalysisSchemes
	cloudsearch:DescribeAvailabilityOptions
	cloudsearch:DescribeDomainEndpointOptions
	cloudsearch:DescribeDomains
	cloudsearch:DescribeExpressions
	cloudsearch:DescribeIndexFields
	cloudsearch:DescribeScalingParameters
	cloudsearch:DescribeServiceAccessPolicies
	cloudsearch:DescribeSuggesters
	cloudsearch:IndexDocuments

Service prefix	Actions
	cloudsearch:ListDomainNames cloudsearch:UpdateAvailabilityOptions cloudsearch:UpdateDomainEndpointOptions cloudsearch:UpdateScalingParameters cloudsearch:UpdateServiceAccessPolicies

Service prefix	Actions
cloudtrail	cloudtrail:CancelQuery cloudtrail:CreateChannel cloudtrail:CreateDashboard cloudtrail:CreateEventDataStore cloudtrail:CreateTrail cloudtrail:DeleteChannel cloudtrail:DeleteDashboard cloudtrail:DeleteEventDataStore cloudtrail:DeleteResourcePolicy cloudtrail:DeleteTrail cloudtrail:DeregisterOrganizationDelegatedAdmin cloudtrail:DescribeQuery cloudtrail:DescribeTrails cloudtrail:DisableFederation cloudtrail:GenerateQuery cloudtrail:GetChannel cloudtrail:GetDashboard cloudtrail:GetEventConfiguration cloudtrail:GetEventDataStore cloudtrail:GetEventDataStoreData cloudtrail:GetEventSelectors

Service prefix	Actions
	cloudtrail:GetImport cloudtrail:GetInsightSelectors cloudtrail:GetResourcePolicy cloudtrail:GetTrail cloudtrail:GetTrailStatus cloudtrail:ListChannels cloudtrail:ListDashboards cloudtrail:ListEventDataStores cloudtrail:ListImportFailures cloudtrail:ListImports cloudtrail:ListInsightsData cloudtrail:ListPublicKeys cloudtrail:ListQueries cloudtrail:ListTrails cloudtrail:LookupEvents cloudtrail:PutEventConfiguration cloudtrail:PutEventSelectors cloudtrail:PutInsightSelectors cloudtrail:PutResourcePolicy cloudtrail:RegisterOrganizationDelegatedAdmin cloudtrail:RestoreEventDataStore

Service prefix	Actions
	cloudtrail:SearchSampleQueries cloudtrail:StartEventDataStoreIngestion cloudtrail:StartImport cloudtrail:StartLogging cloudtrail:StartQuery cloudtrail:StopEventDataStoreIngestion cloudtrail:StopImport cloudtrail:StopLogging cloudtrail:UpdateChannel cloudtrail:UpdateDashboard cloudtrail:UpdateEventDataStore cloudtrail:UpdateTrail

Service prefix	Actions
cloudwatch	cloudwatch:DeleteAlarmMuteRule
	cloudwatch:DeleteAlarms
	cloudwatch:DeleteAnomalyDetector
	cloudwatch:DeleteDashboards
	cloudwatch:DeleteInsightRules
	cloudwatch:DeleteMetricStream
	cloudwatch:DescribeAlarmHistory
	cloudwatch:DescribeAlarms
	cloudwatch:DescribeAlarmsForMetric
	cloudwatch:DescribeAnomalyDetectors
	cloudwatch:DescribeInsightRules
	cloudwatch:DisableAlarmActions
	cloudwatch:DisableInsightRules
	cloudwatch:EnableAlarmActions
	cloudwatch:EnableInsightRules
	cloudwatch:GetAlarmMuteRule
	cloudwatch:GetDashboard
	cloudwatch:GetDataset
	cloudwatch:GetInsightRuleReport
	cloudwatch:GetMetricStatistics
	cloudwatch:GetMetricStream

Service prefix	Actions
	cloudwatch:ListAlarmMuteRules
	cloudwatch:ListDashboards
	cloudwatch:ListManagedInsightRules
	cloudwatch:ListMetricStreams
	cloudwatch:PutAlarmMuteRule
	cloudwatch:PutAnomalyDetector
	cloudwatch:PutCompositeAlarm
	cloudwatch:PutDashboard
	cloudwatch:PutInsightRule
	cloudwatch:PutLogAlarm
	cloudwatch:PutManagedInsightRules
	cloudwatch:PutMetricAlarm
	cloudwatch:PutMetricStream
	cloudwatch:SetAlarmState
	cloudwatch:StartMetricStreams
	cloudwatch:StopMetricStreams

Service prefix	Actions
codeartifact	codeartifact:AssociateExternalConnection codeartifact:CopyPackageVersions codeartifact:CreateDomain codeartifact:CreateRepository codeartifact:DeleteDomain codeartifact:DeleteDomainPermissionsPolicy codeartifact:DeletePackage codeartifact:DeletePackageVersions codeartifact:DeleteRepository codeartifact:DeleteRepositoryPermissionsPolicy codeartifact:DescribeDomain codeartifact:DescribePackage codeartifact:DescribePackageVersion codeartifact:DescribeRepository codeartifact:DisassociateExternalConnection codeartifact:DisposePackageVersions codeartifact:GetAssociatedPackageGroup codeartifact:GetAuthorizationToken codeartifact:GetDomainPermissionsPolicy codeartifact:GetPackageVersionAsset codeartifact:GetPackageVersionReadme

Service prefix	Actions
	codeartifact:GetRepositoryEndpoint codeartifact:GetRepositoryPermissionsPolicy codeartifact:ListDomains codeartifact:ListPackageGroups codeartifact:ListPackageVersionAssets codeartifact:ListPackageVersionDependencies codeartifact:ListPackageVersions codeartifact:ListPackages codeartifact:ListRepositories codeartifact:ListRepositoriesInDomain codeartifact:PublishPackageVersion codeartifact:PutDomainPermissionsPolicy codeartifact:PutPackageMetadata codeartifact:PutPackageOriginConfiguration codeartifact:PutRepositoryPermissionsPolicy codeartifact:ReadFromRepository codeartifact:UpdatePackageVersionsStatus codeartifact:UpdateRepository

Service prefix	Actions
codedeploy	codedeploy:BatchGetApplicationRevisions
	codedeploy:BatchGetApplications
	codedeploy:BatchGetDeploymentGroups
	codedeploy:BatchGetDeploymentInstances
	codedeploy:BatchGetDeploymentTargets
	codedeploy:BatchGetDeployments
	codedeploy:BatchGetOnPremisesInstances
	codedeploy:ContinueDeployment
	codedeploy:CreateApplication
	codedeploy:CreateDeployment
	codedeploy:CreateDeploymentConfig
	codedeploy:CreateDeploymentGroup
	codedeploy:DeleteApplication
	codedeploy:DeleteDeploymentConfig
	codedeploy:DeleteDeploymentGroup
	codedeploy:DeleteGitHubAccountToken
	codedeploy:DeleteResourcesByExternalId
	codedeploy:DeregisterOnPremisesInstance
	codedeploy:GetApplication
	codedeploy:GetApplicationRevision
	codedeploy:GetDeployment

Service prefix	Actions
	codedeploy:GetDeploymentConfig
	codedeploy:GetDeploymentGroup
	codedeploy:GetDeploymentInstance
	codedeploy:GetDeploymentTarget
	codedeploy:GetOnPremisesInstance
	codedeploy:ListApplicationRevisions
	codedeploy:ListApplications
	codedeploy:ListDeploymentConfigs
	codedeploy:ListDeploymentGroups
	codedeploy:ListDeploymentInstances
	codedeploy:ListDeploymentTargets
	codedeploy:ListDeployments
	codedeploy:ListGitHubAccountTokenNames
	codedeploy:ListOnPremisesInstances
	codedeploy:PutLifecycleEventHookExecutionStatus
	codedeploy:RegisterApplicationRevision
	codedeploy:RegisterOnPremisesInstance
	codedeploy:SkipWaitTimeForInstanceTermination
	codedeploy:StopDeployment
	codedeploy:UpdateApplication
	codedeploy:UpdateDeploymentGroup

Service prefix	Actions
codeguru-profiler	codeguru-profiler:AddNotificationChannels codeguru-profiler:BatchGetFrameMetricData codeguru-profiler:CreateProfilingGroup codeguru-profiler:DeleteProfilingGroup codeguru-profiler:DescribeProfilingGroup codeguru-profiler:GetFindingsReportAccountSummary codeguru-profiler:GetNotificationConfiguration codeguru-profiler:GetPolicy codeguru-profiler:GetProfile codeguru-profiler:GetRecommendations codeguru-profiler:ListFindingsReports codeguru-profiler:ListProfileTimes codeguru-profiler:ListProfilingGroups codeguru-profiler:PutPermission codeguru-profiler:RemoveNotificationChannel codeguru-profiler:RemovePermission codeguru-profiler:SubmitFeedback codeguru-profiler:UpdateProfilingGroup

Service prefix	Actions
codeguru-reviewer	codeguru-reviewer:AssociateRepository codeguru-reviewer:CreateCodeReview codeguru-reviewer:DescribeCodeReview codeguru-reviewer:DescribeRecommendationFeedback codeguru-reviewer:DescribeRepositoryAssociation codeguru-reviewer:DisassociateRepository codeguru-reviewer:ListCodeReviews codeguru-reviewer:ListRecommendationFeedback codeguru-reviewer:ListRecommendations codeguru-reviewer:ListRepositoryAssociations codeguru-reviewer:PutRecommendationFeedback

Service prefix	Actions
codepipeline	codepipeline:AcknowledgeJob codepipeline:AcknowledgeThirdPartyJob codepipeline:CreateCustomActionType codepipeline:CreatePipeline codepipeline:DeleteCustomActionType codepipeline:DeletePipeline codepipeline:DeleteWebhook codepipeline:DeregisterWebhookWithThirdParty codepipeline:GetActionType codepipeline:GetJobDetails codepipeline:GetPipeline codepipeline:GetPipelineExecution codepipeline:GetPipelineState codepipeline:GetThirdPartyJobDetails codepipeline:ListActionExecutions codepipeline:ListActionTypes codepipeline:ListPipelineExecutions codepipeline:ListPipelines codepipeline:ListRuleExecutions codepipeline:ListRuleTypes codepipeline:ListWebhooks

Service prefix	Actions
	codepipeline:OverrideStageCondition codepipeline:PollForJobs codepipeline:PollForThirdPartyJobs codepipeline:PutActionRevision codepipeline:PutApprovalResult codepipeline:PutJobFailureResult codepipeline:PutJobSuccessResult codepipeline:PutThirdPartyJobFailureResult codepipeline:PutThirdPartyJobSuccessResult codepipeline:PutWebhook codepipeline:RegisterWebhookWithThirdParty codepipeline:RollbackStage codepipeline:StartPipelineExecution codepipeline:StopPipelineExecution codepipeline:UpdateActionType codepipeline:UpdatePipeline

Service prefix	Actions
codestar	codestar:AssociateTeamMember
	codestar:CreateProject
	codestar:CreateUserProfile
	codestar:DeleteProject
	codestar:DeleteUserProfile
	codestar:DescribeProject
	codestar:DescribeUserProfile
	codestar:DisassociateTeamMember
	codestar:ListProjects
	codestar:ListResources
	codestar:ListTeamMembers
	codestar:ListUserProfiles
	codestar:UpdateProject
	codestar:UpdateTeamMember
	codestar:UpdateUserProfile

Service prefix	Actions
codestar-notifications	codestar-notifications:CreateNotificationRule codestar-notifications:DeleteNotificationRule codestar-notifications:DeleteTarget codestar-notifications:DescribeNotificationRule codestar-notifications:ListEventTypes codestar-notifications:ListNotificationRules codestar-notifications:ListTargets codestar-notifications:Subscribe codestar-notifications:Unsubscribe codestar-notifications:UpdateNotificationRule

Service prefix	Actions
cognito-identity	cognito-identity:CreateIdentityPool
	cognito-identity:DeleteIdentities
	cognito-identity:DeleteIdentityPool
	cognito-identity:DescribeIdentity
	cognito-identity:DescribeIdentityPool
	cognito-identity:GetIdentityPoolRoles
	cognito-identity:ListIdentities
	cognito-identity:ListIdentityPools
	cognito-identity:LookupDeveloperIdentity
	cognito-identity:MergeDeveloperIdentities
	cognito-identity:SetIdentityPoolRoles
	cognito-identity:UnlinkDeveloperIdentity
	cognito-identity:UpdateIdentityPool

Service prefix	Actions
cognito-idp	cognito-idp:AddCustomAttributes cognito-idp:AddUserPoolClientSecret cognito-idp:AdminAddUserToGroup cognito-idp:AdminConfirmSignUp cognito-idp:AdminCreateUser cognito-idp:AdminDeleteUser cognito-idp:AdminDeleteUserAttributes cognito-idp:AdminDisableProviderForUser cognito-idp:AdminDisableUser cognito-idp:AdminEnableUser cognito-idp:AdminForgetDevice cognito-idp:AdminGetDevice cognito-idp:AdminGetUser cognito-idp:AdminGetUserAuthFactors cognito-idp:AdminInitiateAuth cognito-idp:AdminLinkProviderForUser cognito-idp:AdminListDevices cognito-idp:AdminListGroupsForUser cognito-idp:AdminListUserAuthEvents cognito-idp:AdminRemoveUserFromGroup cognito-idp:AdminResetUserPassword

Service prefix	Actions
	cognito-idp:AdminRespondToAuthChallenge cognito-idp:AdminSetUserMFAPreference cognito-idp:AdminSetUserPassword cognito-idp:AdminSetUserSettings cognito-idp:AdminUpdateAuthEventFeedback cognito-idp:AdminUpdateDeviceStatus cognito-idp:AdminUpdateUserAttributes cognito-idp:AdminUserGlobalSignOut cognito-idp:AssociateSoftwareToken cognito-idp:ChangePassword cognito-idp:ConfirmDevice cognito-idp:ConfirmForgotPassword cognito-idp:ConfirmSignUp cognito-idp:CreateGroup cognito-idp:CreateIdentityProvider cognito-idp:CreateManagedLoginBranding cognito-idp:CreateResourceServer cognito-idp:CreateTerms cognito-idp:CreateUserImportJob cognito-idp:CreateUserPool cognito-idp:CreateUserPoolClient

Service prefix	Actions
	cognito-idp:CreateUserPoolDomain cognito-idp:CreateUserPoolReplica cognito-idp:DeleteGroup cognito-idp:DeleteIdentityProvider cognito-idp:DeleteManagedLoginBranding cognito-idp:DeleteResourceServer cognito-idp:DeleteTerms cognito-idp:DeleteUser cognito-idp:DeleteUserAttributes cognito-idp:DeleteUserPool cognito-idp:DeleteUserPoolClient cognito-idp:DeleteUserPoolClientSecret cognito-idp:DeleteUserPoolDomain cognito-idp:DeleteUserPoolReplica cognito-idp:DescribeIdentityProvider cognito-idp:DescribeManagedLoginBranding cognito-idp:DescribeManagedLoginBrandingByClient cognito-idp:DescribeResourceServer cognito-idp:DescribeRiskConfiguration cognito-idp:DescribeTerms cognito-idp:DescribeUserImportJob

Service prefix	Actions
	cognito-idp:DescribeUserPool cognito-idp:DescribeUserPoolClient cognito-idp:DescribeUserPoolDomain cognito-idp:ForgetDevice cognito-idp:ForgotPassword cognito-idp:GetCSVHeader cognito-idp:GetDevice cognito-idp:GetGroup cognito-idp:GetIdentityProviderByIdentifier cognito-idp:GetLogDeliveryConfiguration cognito-idp:GetProvisionedLimit cognito-idp:GetSigningCertificate cognito-idp:GetUICustomization cognito-idp:GetUser cognito-idp:GetUserAttributeVerificationCode cognito-idp:GetUserPoolMfaConfig cognito-idp:GlobalSignOut cognito-idp:InitiateAuth cognito-idp:ListDevices cognito-idp:ListGroups cognito-idp:ListIdentityProviders

Service prefix	Actions
	cognito-idp:ListResourceServers cognito-idp:ListTerms cognito-idp:ListUserImportJobs cognito-idp:ListUserPoolClientSecrets cognito-idp:ListUserPoolClients cognito-idp:ListUserPoolReplicas cognito-idp:ListUserPools cognito-idp:ListUsers cognito-idp:ListUsersInGroup cognito-idp:ResendConfirmationCode cognito-idp:RespondToAuthChallenge cognito-idp:RevokeToken cognito-idp:SetLogDeliveryConfiguration cognito-idp:SetRiskConfiguration cognito-idp:SetUICustomization cognito-idp:SetUserMFAPreference cognito-idp:SetUserPoolMfaConfig cognito-idp:SetUserSettings cognito-idp:SignUp cognito-idp:StartUserImportJob cognito-idp:StopUserImportJob

Service prefix	Actions
	cognito-idp:UpdateAuthEventFeedback cognito-idp:UpdateDeviceStatus cognito-idp:UpdateGroup cognito-idp:UpdateIdentityProvider cognito-idp:UpdateProvisionedLimit cognito-idp:UpdateResourceServer cognito-idp:UpdateTerms cognito-idp:UpdateUserAttributes cognito-idp:UpdateUserPool cognito-idp:UpdateUserPoolClient cognito-idp:UpdateUserPoolDomain cognito-idp:UpdateUserPoolReplica cognito-idp:VerifySoftwareToken cognito-idp:VerifyUserAttribute

Service prefix	Actions
cognito-sync	cognito-sync:BulkPublish
	cognito-sync:DeleteDataset
	cognito-sync:DescribeDataset
	cognito-sync:DescribeIdentityPoolUsage
	cognito-sync:DescribeIdentityUsage
	cognito-sync:GetBulkPublishDetails
	cognito-sync:GetCognitoEvents
	cognito-sync:GetIdentityPoolConfiguration
	cognito-sync:ListDatasets
	cognito-sync:ListIdentityPoolUsage
	cognito-sync:ListRecords
	cognito-sync:RegisterDevice
	cognito-sync:SetCognitoEvents
	cognito-sync:SetIdentityPoolConfiguration
	cognito-sync:SubscribeToDataset
	cognito-sync:UnsubscribeFromDataset
	cognito-sync:UpdateRecords

Service prefix	Actions
comprehendmedical	comprehendmedical:DescribeEntitiesDetectionV2Job comprehendmedical:DescribeICD10CMInferenceJob comprehendmedical:DescribePHIDetectionJob comprehendmedical:DescribeRxNormInferenceJob comprehendmedical:DescribeSNOMEDCTInferenceJob comprehendmedical:DetectEntitiesV2 comprehendmedical:DetectPHI comprehendmedical:InferICD10CM comprehendmedical:InferRxNorm comprehendmedical:InferSNOMEDCT comprehendmedical:ListEntitiesDetectionV2Jobs comprehendmedical:ListICD10CMInferenceJobs comprehendmedical:ListPHIDetectionJobs comprehendmedical:ListRxNormInferenceJobs comprehendmedical:ListSNOMEDCTInferenceJobs comprehendmedical:StartEntitiesDetectionV2Job comprehendmedical:StartICD10CMInferenceJob comprehendmedical:StartPHIDetectionJob comprehendmedical:StartRxNormInferenceJob comprehendmedical:StartSNOMEDCTInferenceJob comprehendmedical:StopEntitiesDetectionV2Job

Service prefix	Actions
	comprehendmedical:StopICD10CMInferenceJob comprehendmedical:StopPHIDetectionJob comprehendmedical:StopRxNormInferenceJob comprehendmedical:StopSNOMEDCTInferenceJob

Service prefix	Actions
compute-optimizer	compute-optimizer:DeleteRecommendationPreferences
	compute-optimizer:DescribeRecommendationExportJobs
	compute-optimizer:ExportAutoScalingGroupRecommendations
	compute-optimizer:ExportEBSVolumeRecommendations
	compute-optimizer:ExportEC2InstanceRecommendations
	compute-optimizer:ExportECSServiceRecommendations
	compute-optimizer:ExportIdleRecommendations
	compute-optimizer:ExportLambdaFunctionRecommendations
	compute-optimizer:ExportLicenseRecommendations
	compute-optimizer:ExportRDSDatabaseRecommendations
	compute-optimizer:GetEC2RecommendationProjectedMetrics
	compute-optimizer:GetECSServiceRecommendationProjected Metrics
	compute-optimizer:GetEffectiveRecommendationPreferences
	compute-optimizer:GetEnrollmentStatus
	compute-optimizer:GetEnrollmentStatusesForOrganization
	compute-optimizer:GetRDSDatabaseRecommendationProjecte dMetrics
	compute-optimizer:GetRecommendationPreferences
	compute-optimizer:GetRecommendationSummaries
	compute-optimizer:PutRecommendationPreferences

Service prefix	Actions
	compute-optimizer:UpdateEnrollmentStatus

Service prefix	Actions
config	config:BatchGetResourceConfig config:DeleteAggregationAuthorization config:DeleteConfigRule config:DeleteConfigurationAggregator config:DeleteConfigurationRecorder config:DeleteConformancePack config:DeleteDeliveryChannel config:DeleteEvaluationResults config:DeleteOrganizationConfigRule config:DeleteOrganizationConformancePack config:DeletePendingAggregationRequest config:DeleteRemediationConfiguration config:DeleteRemediationExceptions config:DeleteResourceConfig config:DeleteRetentionConfiguration config:DeleteStoredQuery config:DeliverConfigSnapshot config:DescribeAggregateComplianceByConfigRules config:DescribeAggregateComplianceByConformancePacks config:DescribeAggregationAuthorizations config:DescribeComplianceByConfigRule

Service prefix	Actions
	config:DescribeComplianceByResource config:DescribeConfigRuleEvaluationStatus config:DescribeConfigRules config:DescribeConfigurationAggregatorSourcesStatus config:DescribeConfigurationAggregators config:DescribeConfigurationRecorderStatus config:DescribeConfigurationRecorders config:DescribeConformancePackCompliance config:DescribeConformancePackStatus config:DescribeConformancePacks config:DescribeDeliveryChannelStatus config:DescribeDeliveryChannels config:DescribeOrganizationConfigRuleStatuses config:DescribeOrganizationConfigRules config:DescribeOrganizationConformancePackStatuses config:DescribeOrganizationConformancePacks config:DescribePendingAggregationRequests config:DescribeRemediationConfigurations config:DescribeRemediationExceptions config:DescribeRemediationExecutionStatus config:DescribeRetentionConfigurations

Service prefix	Actions
	config:GetComplianceDetailsByConfigRule config:GetComplianceDetailsByResource config:GetComplianceSummaryByConfigRule config:GetComplianceSummaryByResourceType config:GetConformancePackComplianceDetails config:GetConformancePackComplianceSummary config:GetCustomRulePolicy config:GetDiscoveredResourceCounts config:GetOrganizationConfigRuleDetailedStatus config:GetOrganizationConformancePackDetailedStatus config:GetOrganizationCustomRulePolicy config:GetResourceConfigHistory config:GetResourceEvaluationSummary config:GetStoredQuery config:ListConfigurationRecorders config:ListConformancePackComplianceScores config:ListConnectors config:ListDiscoveredResources config:ListResourceEvaluations config:ListStoredQueries config:PutConfigRule

Service prefix	Actions
	config:PutConfigurationAggregator config:PutConfigurationRecorder config:PutConformancePack config:PutConnector config:PutDeliveryChannel config:PutEvaluations config:PutExternalEvaluation config:PutOrganizationConfigRule config:PutOrganizationConformancePack config:PutRemediationConfigurations config:PutRemediationExceptions config:PutResourceConfig config:PutRetentionConfiguration config:PutStoredQuery config:SelectResourceConfig config:StartConfigRulesEvaluation config:StartConfigurationRecorder config:StartRemediationExecution config:StartResourceEvaluation config:StopConfigurationRecorder

Service prefix	Actions
connect	connect:ActivateEvaluationForm
	connect:AssociateAnalyticsDataSet
	connect:AssociateApprovedOrigin
	connect:AssociateBot
	connect:AssociateContactWithUser
	connect:AssociateDefaultVocabulary
	connect:AssociateEmailAddressAlias
	connect:AssociateFlow
	connect:AssociateInstanceStorageConfig
	connect:AssociateLambdaFunction
	connect:AssociateLexBot
	connect:AssociatePhoneNumberContactFlow
	connect:AssociateQueueEmailAddresses
	connect:AssociateQueueQuickConnects
	connect:AssociateRoutingProfileQueues
	connect:AssociateSecurityKey
	connect:AssociateUserProficiencies
	connect:BatchAssociateAnalyticsDataSet
	connect:BatchCreateDataTableValue
	connect:BatchDeleteDataTableValue
	connect:BatchDescribeDataTableValue

Service prefix	Actions
	connect:BatchDisassociateAnalyticsDataSet
	connect:BatchGetFlowAssociation
	connect:BatchPutContact
	connect:BatchUpdateDataTableValue
	connect:ClaimPhoneNumber
	connect:CreateAgentStatus
	connect:CreateAttachedFile
	connect:CreateContact
	connect:CreateContactFlow
	connect:CreateContactFlowModule
	connect:CreateContactFlowModuleAlias
	connect:CreateContactFlowModuleVersion
	connect:CreateContactFlowVersion
	connect:CreateDataTable
	connect:CreateDataTableAttribute
	connect:CreateEmailAddress
	connect:CreateEvaluationForm
	connect:CreateHoursOfOperation
	connect:CreateInstance
	connect:CreateIntegrationAssociation
	connect:CreateParticipant

Service prefix	Actions
	connect:CreatePersistentContactAssociation
	connect:CreatePredefinedAttribute
	connect:CreatePrompt
	connect:CreatePushNotificationRegistration
	connect:CreateQueue
	connect:CreateQuickConnect
	connect:CreateRoutingProfile
	connect:CreateRule
	connect:CreateSecurityProfile
	connect:CreateTaskTemplate
	connect:CreateTrafficDistributionGroup
	connect:CreateUseCase
	connect:CreateUser
	connect:CreateUserHierarchyGroup
	connect:CreateView
	connect:CreateViewVersion
	connect:CreateVocabulary
	connect:CreateWorkspace
	connect:DeactivateEvaluationForm
	connect:DeleteContactEvaluation
	connect:DeleteContactFlow

Service prefix	Actions
	connect:DeleteContactFlowModule
	connect:DeleteContactFlowModuleAlias
	connect:DeleteContactFlowModuleVersion
	connect:DeleteContactFlowVersion
	connect:DeleteDataTable
	connect:DeleteDataTableAttribute
	connect:DeleteEmailAddress
	connect:DeleteEvaluationForm
	connect:DeleteHoursOfOperation
	connect:DeleteHoursOfOperationOverride
	connect:DeleteInstance
	connect:DeleteIntegrationAssociation
	connect:DeleteNotification
	connect:DeletePredefinedAttribute
	connect:DeletePrompt
	connect:DeletePushNotificationRegistration
	connect:DeleteQueue
	connect:DeleteQuickConnect
	connect:DeleteRoutingProfile
	connect:DeleteRule
	connect:DeleteSecurityProfile

Service prefix	Actions
	connect:DeleteTaskTemplate
	connect:DeleteTrafficDistributionGroup
	connect:DeleteUseCase
	connect:DeleteUser
	connect:DeleteUserHierarchyGroup
	connect:DeleteView
	connect:DeleteVocabulary
	connect:DeleteWorkspace
	connect:DeleteWorkspaceMedia
	connect:DescribeAttachedFilesConfiguration
	connect:DescribeAuthenticationProfile
	connect:DescribeContactFlowModuleAlias
	connect:DescribeDataTableAttribute
	connect:DescribeHoursOfOperationOverride
	connect:DescribeInstanceAttribute
	connect:DescribeInstanceStorageConfig
	connect:DescribeNotification
	connect:DescribePhoneNumber
	connect:DescribeRule
	connect:DescribeTrafficDistributionGroup
	connect:DescribeUserHierarchyStructure

Service prefix	Actions
	connect:DescribeVocabulary
	connect:DisassociateAnalyticsDataSet
	connect:DisassociateApprovedOrigin
	connect:DisassociateBot
	connect:DisassociateEmailAddressAlias
	connect:DisassociateFlow
	connect:DisassociateInstanceStorageConfig
	connect:DisassociateLambdaFunction
	connect:DisassociateLexBot
	connect:DisassociatePhoneNumberContactFlow
	connect:DisassociateQueueEmailAddresses
	connect:DisassociateQueueQuickConnects
	connect:DisassociateRoutingProfileQueues
	connect:DisassociateSecurityKey
	connect:DisassociateUserProficiencies
	connect:DismissUserContact
	connect:EvaluateDataTableValues
	connect:GetContactAttributes
	connect:GetContactMetrics
	connect:GetCurrentMetricData
	connect:GetCurrentUserData

Service prefix	Actions
	connect:GetEffectiveHoursOfOperations
	connect:GetEvaluationFormValidation
	connect:GetFederationToken
	connect:GetFlowAssociation
	connect:GetMetricData
	connect:GetMetricDataV2
	connect:GetPromptFile
	connect:GetTaskTemplate
	connect:GetTrafficDistribution
	connect:ImportPhoneNumber
	connect:ImportWorkspaceMedia
	connect:ListAnalyticsDataAssociations
	connect:ListAnalyticsDataLakeDataSets
	connect:ListApprovedOrigins
	connect:ListAssociatedContacts
	connect:ListAttachedFilesConfigurations
	connect:ListAuthenticationProfiles
	connect:ListBots
	connect:ListContactEvaluations
	connect:ListContactFlowModuleAliases
	connect:ListContactFlowModuleVersions

Service prefix	Actions
	connect:ListContactFlowModules
	connect:ListContactFlowVersions
	connect:ListContactFlows
	connect:ListContactReferences
	connect:ListDataTableAttributes
	connect:ListDataTablePrimaryValues
	connect:ListDataTableValues
	connect:ListDataTables
	connect:ListDefaultVocabularies
	connect:ListEvaluationFormVersions
	connect:ListEvaluationForms
	connect:ListFlowAssociations
	connect:ListHoursOfOperations
	connect:ListInstanceAttributes
	connect:ListInstanceStorageConfigs
	connect:ListIntegrationAssociations
	connect:ListLambdaFunctions
	connect:ListLexBots
	connect:ListNotifications
	connect:ListPhoneNumbers
	connect:ListPhoneNumbersV2

Service prefix	Actions
	connect:ListPredefinedAttributes
	connect:ListPrompts
	connect:ListQueueEmailAddresses
	connect:ListQueueQuickConnects
	connect:ListQueues
	connect:ListQuickConnects
	connect:ListRealtimeContactAnalysisSegmentsV2
	connect:ListRoutingProfileManualAssignmentQueues
	connect:ListRoutingProfileQueues
	connect:ListRoutingProfiles
	connect:ListRules
	connect:ListSecurityKeys
	connect:ListSecurityProfileApplications
	connect:ListSecurityProfileFlowModules
	connect:ListSecurityProfilePermissions
	connect:ListSecurityProfiles
	connect:ListTaskTemplates
	connect:ListTrafficDistributionGroups
	connect:ListUseCases
	connect:ListUserHierarchyGroups
	connect:ListUserNotifications

Service prefix	Actions
	connect:ListUsers
	connect:ListViewVersions
	connect:ListViews
	connect:ListWorkspaceMedia
	connect:ListWorkspacePages
	connect:ListWorkspaces
	connect:MonitorContact
	connect:PauseContact
	connect:PutUserStatus
	connect:ReleasePhoneNumber
	connect:ReplicateInstance
	connect:ResumeContact
	connect:ResumeContactRecording
	connect:SearchAgentStatuses
	connect:SearchAvailablePhoneNumbers
	connect:SearchContactEvaluations
	connect:SearchContactFlowModules
	connect:SearchContactFlows
	connect:SearchContacts
	connect:SearchDataTables
	connect:SearchEmailAddresses

Service prefix	Actions
	connect:SearchEvaluationForms
	connect:SearchHoursOfOperations
	connect:SearchPredefinedAttributes
	connect:SearchPrompts
	connect:SearchQueues
	connect:SearchQuickConnects
	connect:SearchRoutingProfiles
	connect:SearchRules
	connect:SearchSecurityProfiles
	connect:SearchUserHierarchyGroups
	connect:SearchViews
	connect:SearchVocabularies
	connect:SearchWorkspaceAssociations
	connect:SearchWorkspaces
	connect:SendChatIntegrationEvent
	connect:SendOutboundEmail
	connect:StartChatContact
	connect:StartContactConversationalAnalyticsJob
	connect:StartContactEvaluation
	connect:StartContactMediaProcessing
	connect:StartContactRecording

Service prefix	Actions
	connect:StartContactStreaming
	connect:StartEmailContact
	connect:StartOutboundChatContact
	connect:StartOutboundEmailContact
	connect:StartOutboundVoiceContact
	connect:StartScreenSharing
	connect:StartTaskContact
	connect:StartWebRTCContact
	connect:StopContact
	connect:StopContactMediaProcessing
	connect:StopContactRecording
	connect:StopContactStreaming
	connect:SubmitContactEvaluation
	connect:SuspendContactRecording
	connect:TransferContact
	connect:UpdateAgentStatus
	connect:UpdateAttachedFilesConfiguration
	connect:UpdateAuthenticationProfile
	connect:UpdateContact
	connect:UpdateContactAttributes
	connect:UpdateContactEvaluation

Service prefix	Actions
	connect:UpdateContactFlowContent
	connect:UpdateContactFlowMetadata
	connect:UpdateContactFlowModuleAlias
	connect:UpdateContactFlowModuleContent
	connect:UpdateContactFlowModuleMetadata
	connect:UpdateContactFlowName
	connect:UpdateContactRoutingData
	connect:UpdateContactSchedule
	connect:UpdateDataTableAttribute
	connect:UpdateDataTableMetadata
	connect:UpdateDataTablePrimaryValues
	connect:UpdateEmailAddressMetadata
	connect:UpdateEvaluationForm
	connect:UpdateHoursOfOperation
	connect:UpdateHoursOfOperationOverride
	connect:UpdateInstanceAttribute
	connect:UpdateInstanceStorageConfig
	connect:UpdateNotificationContent
	connect:UpdateParticipantAuthentication
	connect:UpdateParticipantRoleConfig
	connect:UpdatePhoneNumber

Service prefix	Actions
	connect:UpdatePhoneNumberMetadata
	connect:UpdatePredefinedAttribute
	connect:UpdatePrompt
	connect:UpdateQueueHoursOfOperation
	connect:UpdateQueueMaxContacts
	connect:UpdateQueueName
	connect:UpdateQueueOutboundCallerConfig
	connect:UpdateQueueOutboundEmailConfig
	connect:UpdateQueueStatus
	connect:UpdateQuickConnectConfig
	connect:UpdateQuickConnectName
	connect:UpdateRoutingProfileAgentAvailabilityTimer
	connect:UpdateRoutingProfileConcurrency
	connect:UpdateRoutingProfileDefaultOutboundQueue
	connect:UpdateRoutingProfileName
	connect:UpdateRoutingProfileQueues
	connect:UpdateRule
	connect:UpdateSecurityProfile
	connect:UpdateTaskTemplate
	connect:UpdateTrafficDistribution
	connect:UpdateUserHierarchy

Service prefix	Actions
	connect:UpdateUserHierarchyGroupName connect:UpdateUserHierarchyStructure connect:UpdateUserIdentityInfo connect:UpdateUserNotificationStatus connect:UpdateUserPhoneConfig connect:UpdateUserProficiencies connect:UpdateUserRoutingProfile connect:UpdateUserSecurityProfiles connect:UpdateViewContent connect:UpdateViewMetadata connect:UpdateWorkspaceMetadata connect:UpdateWorkspaceTheme connect:UpdateWorkspaceVisibility
cur	cur:DeleteReportDefinition cur:DescribeReportDefinitions cur:ModifyReportDefinition cur:PutReportDefinition

Service prefix	Actions
databrew	databrew:BatchDeleteRecipeVersion
	databrew:CreateDataset
	databrew:CreateProfileJob
	databrew:CreateProject
	databrew:CreateRecipe
	databrew:CreateRecipeJob
	databrew:CreateRuleset
	databrew:CreateSchedule
	databrew:DeleteDataset
	databrew:DeleteJob
	databrew:DeleteProject
	databrew:DeleteRecipeVersion
	databrew:DeleteRuleset
	databrew:DeleteSchedule
	databrew:DescribeDataset
	databrew:DescribeJob
	databrew:DescribeJobRun
	databrew:DescribeProject
	databrew:DescribeRecipe
	databrew:DescribeRuleset
	databrew:DescribeSchedule

Service prefix	Actions
	databrew:ListDatasets
	databrew:ListJobRuns
	databrew:ListJobs
	databrew:ListProjects
	databrew:ListRecipeVersions
	databrew:ListRecipes
	databrew:ListRulesets
	databrew:ListSchedules
	databrew:PublishRecipe
	databrew:SendProjectSessionAction
	databrew:StartJobRun
	databrew:StartProjectSession
	databrew:StopJobRun
	databrew:UpdateDataset
	databrew:UpdateProfileJob
	databrew:UpdateProject
	databrew:UpdateRecipe
	databrew:UpdateRecipeJob
	databrew:UpdateRuleset
	databrew:UpdateSchedule

Service prefix	Actions
dataexchange	dataexchange:AcceptDataGrant dataexchange:CancelJob dataexchange:CreateDataGrant dataexchange:CreateDataSet dataexchange:CreateEventAction dataexchange:CreateJob dataexchange:CreateRevision dataexchange:DeleteAsset dataexchange:DeleteDataGrant dataexchange:DeleteEventAction dataexchange:DeleteRevision dataexchange:GetDataGrant dataexchange:GetEventAction dataexchange:GetJob dataexchange:GetReceivedDataGrant dataexchange:ListDataGrants dataexchange:ListDataSetRevisions dataexchange:ListDataSets dataexchange:ListEventActions dataexchange:ListJobs dataexchange:ListReceivedDataGrants

Service prefix	Actions
	dataexchange:ListRevisionAssets dataexchange:RevokeRevision dataexchange:SendDataSetNotification dataexchange:StartJob dataexchange:UpdateAsset dataexchange:UpdateDataSet dataexchange:UpdateEventAction dataexchange:UpdateRevision

Service prefix	Actions
datapipeline	datapipeline:ActivatePipeline datapipeline:CreatePipeline datapipeline:DeactivatePipeline datapipeline:DeletePipeline datapipeline:DescribeObjects datapipeline:DescribePipelines datapipeline:EvaluateExpression datapipeline:GetPipelineDefinition datapipeline:ListPipelines datapipeline:PollForTask datapipeline:PutPipelineDefinition datapipeline:QueryObjects datapipeline:ReportTaskProgress datapipeline:ReportTaskRunnerHeartbeat datapipeline:SetStatus datapipeline:SetTaskStatus datapipeline:ValidatePipelineDefinition

Service prefix	Actions
dax	dax:CreateCluster dax:DecreaseReplicationFactor dax:DeleteCluster dax:DeleteParameterGroup dax:DeleteSubnetGroup dax:DescribeClusters dax:DescribeDefaultParameters dax:DescribeEvents dax:DescribeParameterGroups dax:DescribeParameters dax:DescribeSubnetGroups dax:IncreaseReplicationFactor dax:RebootNode dax:UpdateCluster dax:UpdateParameterGroup dax:UpdateSubnetGroup

Service prefix	Actions
devicefarm	devicefarm:CreateDevicePool
	devicefarm:CreateInstanceProfile
	devicefarm:CreateNetworkProfile
	devicefarm:CreateProject
	devicefarm:CreateRemoteAccessSession
	devicefarm:CreateTestGridProject
	devicefarm:CreateTestGridUrl
	devicefarm:CreateUpload
	devicefarm:CreateVPCEConfiguration
	devicefarm:DeleteDevicePool
	devicefarm:DeleteInstanceProfile
	devicefarm:DeleteNetworkProfile
	devicefarm:DeleteProject
	devicefarm:DeleteRemoteAccessSession
	devicefarm:DeleteRun
	devicefarm:DeleteTestGridProject
	devicefarm:DeleteUpload
	devicefarm:DeleteVPCEConfiguration
	devicefarm:GetAccountSettings
	devicefarm:GetDevice
	devicefarm:GetDeviceInstance

Service prefix	Actions
	devicefarm:GetDevicePool
	devicefarm:GetDevicePoolCompatibility
	devicefarm:GetInstanceProfile
	devicefarm:GetJob
	devicefarm:GetNetworkProfile
	devicefarm:GetOfferingStatus
	devicefarm:GetProject
	devicefarm:GetRemoteAccessSession
	devicefarm:GetRun
	devicefarm:GetSuite
	devicefarm:GetTest
	devicefarm:GetTestGridProject
	devicefarm:GetTestGridSession
	devicefarm:GetUpload
	devicefarm:GetVPCEConfiguration
	devicefarm:ListArtifacts
	devicefarm:ListDeviceInstances
	devicefarm:ListDevicePools
	devicefarm:ListDevices
	devicefarm:ListInstanceProfiles
	devicefarm:ListJobs

Service prefix	Actions
	devicefarm:ListNetworkProfiles
	devicefarm:ListOfferingPromotions
	devicefarm:ListOfferingTransactions
	devicefarm:ListOfferings
	devicefarm:ListProjects
	devicefarm:ListRemoteAccessSessions
	devicefarm:ListRuns
	devicefarm:ListSamples
	devicefarm:ListSuites
	devicefarm:ListTestGridProjects
	devicefarm:ListTestGridSessionActions
	devicefarm:ListTestGridSessionArtifacts
	devicefarm:ListTestGridSessions
	devicefarm:ListTests
	devicefarm:ListUniqueProblems
	devicefarm:ListUploads
	devicefarm:ListVPCEConfigurations
	devicefarm:PurchaseOffering
	devicefarm:RenewOffering
	devicefarm:ScheduleRun
	devicefarm:StopJob

Service prefix	Actions
	devicefarm:StopRemoteAccessSession devicefarm:StopRun devicefarm:UpdateDeviceInstance devicefarm:UpdateDevicePool devicefarm:UpdateInstanceProfile devicefarm:UpdateNetworkProfile devicefarm:UpdateProject devicefarm:UpdateTestGridProject devicefarm:UpdateUpload devicefarm:UpdateVPCEConfiguration

Service prefix	Actions
devops-guru	devops-guru:AddNotificationChannel devops-guru:DeleteInsight devops-guru:DescribeAccountHealth devops-guru:DescribeAccountOverview devops-guru:DescribeAnomaly devops-guru:DescribeEventSourcesConfig devops-guru:DescribeFeedback devops-guru:DescribeInsight devops-guru:DescribeOrganizationHealth devops-guru:DescribeOrganizationOverview devops-guru:DescribeOrganizationResourceCollectionHealth devops-guru:DescribeResourceCollectionHealth devops-guru:DescribeServiceIntegration devops-guru:GetCostEstimation devops-guru:GetResourceCollection devops-guru:ListAnomaliesForInsight devops-guru:ListAnomalousLogGroups devops-guru:ListEvents devops-guru:ListInsights devops-guru:ListMonitoredResources devops-guru:ListNotificationChannels

Service prefix	Actions
	devops-guru:ListOrganizationInsights devops-guru:ListRecommendations devops-guru:PutFeedback devops-guru:RemoveNotificationChannel devops-guru:SearchInsights devops-guru:SearchOrganizationInsights devops-guru:StartCostEstimation devops-guru:UpdateEventSourcesConfig devops-guru:UpdateResourceCollection devops-guru:UpdateServiceIntegration

Service prefix	Actions
directconnect	directconnect:AcceptDirectConnectGatewayAssociationProposal
	directconnect:AllocateConnectionOnInterconnect
	directconnect:AllocateHostedConnection
	directconnect:AllocatePrivateVirtualInterface
	directconnect:AllocatePublicVirtualInterface
	directconnect:AllocateTransitVirtualInterface
	directconnect:AssociateConnectionWithLag
	directconnect:AssociateHostedConnection
	directconnect:AssociateMacSecKey
	directconnect:AssociateVirtualInterface
	directconnect:ConfirmConnection
	directconnect:ConfirmCustomerAgreement
	directconnect:ConfirmPrivateVirtualInterface
	directconnect:ConfirmPublicVirtualInterface
	directconnect:ConfirmTransitVirtualInterface
	directconnect:CreateBGPPeer
	directconnect:CreateConnection
	directconnect:CreateDirectConnectGateway
	directconnect:CreateDirectConnectGatewayAssociation
	directconnect:CreateDirectConnectGatewayAssociationProposal
	directconnect:CreateInterconnect

Service prefix	Actions
	directconnect:CreateLag
	directconnect:CreatePrivateVirtualInterface
	directconnect:CreatePublicVirtualInterface
	directconnect:CreateTransitVirtualInterface
	directconnect:DeleteBGPPeer
	directconnect:DeleteConnection
	directconnect:DeleteDirectConnectGateway
	directconnect:DeleteDirectConnectGatewayAssociation
	directconnect:DeleteDirectConnectGatewayAssociationProposal
	directconnect:DeleteInterconnect
	directconnect:DeleteLag
	directconnect:DeleteVirtualInterface
	directconnect:DescribeConnectionLoa
	directconnect:DescribeConnections
	directconnect:DescribeConnectionsOnInterconnect
	directconnect:DescribeCustomerMetadata
	directconnect:DescribeDirectConnectGatewayAssociationProposals
	directconnect:DescribeDirectConnectGatewayAssociations
	directconnect:DescribeDirectConnectGatewayAttachments
	directconnect:DescribeDirectConnectGateways
	directconnect:DescribeHostedConnections

Service prefix	Actions
	directconnect:DescribeInterconnectLoa
	directconnect:DescribeInterconnects
	directconnect:DescribeLags
	directconnect:DescribeLoa
	directconnect:DescribeLocations
	directconnect:DescribeRouterConfiguration
	directconnect:DescribeVirtualGateways
	directconnect:DescribeVirtualInterfaces
	directconnect:DisassociateConnectionFromLag
	directconnect:DisassociateMacSecKey
	directconnect:ListVirtualInterfaceTestHistory
	directconnect:StartBgpFailoverTest
	directconnect:StopBgpFailoverTest
	directconnect:UpdateConnection
	directconnect:UpdateDirectConnectGateway
	directconnect:UpdateDirectConnectGatewayAssociation
	directconnect:UpdateLag
	directconnect:UpdateVirtualInterfaceAttributes

Service prefix	Actions
dlm	dlm:CreateLifecyclePolicy dlm:DeleteLifecyclePolicy dlm:GetLifecyclePolicies dlm:GetLifecyclePolicy dlm:UpdateLifecyclePolicy

Service prefix	Actions
dms	dms:ApplyPendingMaintenanceAction
	dms:AssociateExtensionPack
	dms:BatchStartRecommendations
	dms:CancelMetadataModelCreation
	dms:CancelReplicationTaskAssessmentRun
	dms:CreateDataProvider
	dms:CreateEndpoint
	dms:CreateEventSubscription
	dms:CreateInstanceProfile
	dms:CreateMigrationProject
	dms:CreateReplicationConfig
	dms:CreateReplicationInstance
	dms:CreateReplicationSubnetGroup
	dms:CreateReplicationTask
	dms:DeleteCertificate
	dms:DeleteConnection
	dms:DeleteDataMigration
	dms:DeleteDataProvider
	dms:DeleteEndpoint
	dms:DeleteEventSubscription
	dms:DeleteFleetAdvisorCollector

Service prefix	Actions
	dms:DeleteFleetAdvisorDatabases
	dms:DeleteInstanceProfile
	dms:DeleteMigrationProject
	dms:DeleteReplicationConfig
	dms:DeleteReplicationInstance
	dms:DeleteReplicationSubnetGroup
	dms:DeleteReplicationTask
	dms:DeleteReplicationTaskAssessmentRun
	dms:DescribeAccountAttributes
	dms:DescribeApplicableIndividualAssessments
	dms:DescribeCertificates
	dms:DescribeConnections
	dms:DescribeDataMigrations
	dms:DescribeEndpointSettings
	dms:DescribeEndpointTypes
	dms:DescribeEndpoints
	dms:DescribeEngineVersions
	dms:DescribeEventCategories
	dms:DescribeEventSubscriptions
	dms:DescribeEvents
	dms:DescribeFleetAdvisorCollectors

Service prefix	Actions
	dms:DescribeFleetAdvisorDatabases
	dms:DescribeFleetAdvisorLsaAnalysis
	dms:DescribeFleetAdvisorSchemaObjectSummary
	dms:DescribeFleetAdvisorSchemas
	dms:DescribeMetadataModel
	dms:DescribeMetadataModelChildren
	dms:DescribeMetadataModelCreations
	dms:DescribeMetadataModelImports
	dms:DescribeOrderableReplicationInstances
	dms:DescribePendingMaintenanceActions
	dms:DescribeRecommendationLimitations
	dms:DescribeRecommendations
	dms:DescribeRefreshSchemasStatus
	dms:DescribeReplicationConfigs
	dms:DescribeReplicationInstanceTaskLogs
	dms:DescribeReplicationInstances
	dms:DescribeReplicationSubnetGroups
	dms:DescribeReplicationTableStatistics
	dms:DescribeReplicationTaskAssessmentResults
	dms:DescribeReplicationTaskAssessmentRuns
	dms:DescribeReplicationTaskIndividualAssessments

Service prefix	Actions
	dms:DescribeReplicationTasks
	dms:DescribeReplications
	dms:DescribeSchemas
	dms:DescribeTableStatistics
	dms:ExportMetadataModelAssessment
	dms:ImportCertificate
	dms:ListDataProviders
	dms:ListExtensionPacks
	dms:ListInstanceProfiles
	dms:ListMetadataModelAssessments
	dms:ListMetadataModelConversions
	dms:ListMetadataModelExports
	dms:ListMigrationProjects
	dms:ModifyDataMigration
	dms:ModifyEndpoint
	dms:ModifyEventSubscription
	dms:ModifyReplicationConfig
	dms:ModifyReplicationInstance
	dms:ModifyReplicationSubnetGroup
	dms:ModifyReplicationTask
	dms:MoveReplicationTask

Service prefix	Actions
	dms:RebootReplicationInstance
	dms:RefreshSchemas
	dms:ReloadReplicationTables
	dms:ReloadTables
	dms:RunFleetAdvisorLsaAnalysis
	dms:StartMetadataModelAssessment
	dms:StartMetadataModelConversion
	dms:StartMetadataModelCreation
	dms:StartMetadataModelExportAsScripts
	dms:StartMetadataModelExportToTarget
	dms:StartRecommendations
	dms:StartReplication
	dms:StartReplicationTask
	dms:StartReplicationTaskAssessment
	dms:StopDataMigration
	dms:StopReplicationTask
	dms:TestConnection
	dms:UpdateConversionConfiguration
	dms:UpdateDataProvider
	dms:UpdateInstanceProfile
	dms:UpdateMigrationProject

Service prefix	Actions
	dms:UpdateSubscriptionsToEventBridge
docdb-elastic	docdb-elastic:ApplyPendingMaintenanceAction docdb-elastic:CopyClusterSnapshot docdb-elastic:DeleteCluster docdb-elastic:DeleteClusterSnapshot docdb-elastic:GetCluster docdb-elastic:GetClusterSnapshot docdb-elastic:GetPendingMaintenanceAction docdb-elastic:ListClusterSnapshots docdb-elastic:ListClusters docdb-elastic:ListPendingMaintenanceActions docdb-elastic:RestoreClusterFromSnapshot docdb-elastic:StartCluster docdb-elastic:StopCluster docdb-elastic:UpdateCluster

Service prefix	Actions
dynamodb	dynamodb:CreateBackup
	dynamodb:CreateGlobalTable
	dynamodb:CreateTable
	dynamodb:DeleteBackup
	dynamodb:DeleteTable
	dynamodb:DescribeBackup
	dynamodb:DescribeContinuousBackups
	dynamodb:DescribeContributorInsights
	dynamodb:DescribeEndpoints
	dynamodb:DescribeExport
	dynamodb:DescribeGlobalTable
	dynamodb:DescribeGlobalTableSettings
	dynamodb:DescribeImport
	dynamodb:DescribeKinesisStreamingDestination
	dynamodb:DescribeLimits
	dynamodb:DescribeStream
	dynamodb:DescribeTable
	dynamodb:DescribeTableReplicaAutoScaling
	dynamodb:DescribeTimeToLive
	dynamodb:DisableKinesisStreamingDestination
	dynamodb:EnableKinesisStreamingDestination

Service prefix	Actions
	dynamodb:ExportTableToPointInTime
	dynamodb:GetResourcePolicy
	dynamodb:ImportTable
	dynamodb:ListBackups
	dynamodb:ListContributorInsights
	dynamodb:ListExports
	dynamodb:ListGlobalTables
	dynamodb:ListImports
	dynamodb:ListStreams
	dynamodb:ListTables
	dynamodb:RestoreTableFromBackup
	dynamodb:RestoreTableToPointInTime
	dynamodb:UpdateContinuousBackups
	dynamodb:UpdateContributorInsights
	dynamodb:UpdateGlobalTable
	dynamodb:UpdateGlobalTableSettings
	dynamodb:UpdateKinesisStreamingDestination
	dynamodb:UpdateTable
	dynamodb:UpdateTableReplicaAutoScaling
	dynamodb:UpdateTimeToLive

Service prefix	Actions
ebs	ebs:CompleteSnapshot ebs:StartSnapshot

Service prefix	Actions
ec2	ec2:AcceptAddressTransfer
	ec2:AcceptCapacityReservationBillingOwnership
	ec2:AcceptReservedInstancesExchangeQuote
	ec2:AcceptTransitGatewayClientVpnAttachment
	ec2:AcceptTransitGatewayMulticastDomainAssociations
	ec2:AcceptTransitGatewayPeeringAttachment
	ec2:AcceptTransitGatewayVpcAttachment
	ec2:AcceptVpcEndpointConnections
	ec2:AcceptVpcPeeringConnection
	ec2:AdvertiseByoipCidr
	ec2:AllocateAddress
	ec2:AllocateHosts
	ec2:AllocateIpamPoolCidr
	ec2:ApplySecurityGroupsToClientVpnTargetNetwork
	ec2:AssignIpv6Addresses
	ec2:AssignPrivateIpAddresses
	ec2:AssignPrivateNatGatewayAddress
	ec2:AssociateAddress
	ec2:AssociateCapacityReservationBillingOwner
	ec2:AssociateClientVpnTargetNetwork
	ec2:AssociateDhcpOptions

Service prefix	Actions
	ec2:AssociateEnclaveCertificateIamRole
	ec2:AssociateIamInstanceProfile
	ec2:AssociateInstanceEventWindow
	ec2:AssociateIpamByoasn
	ec2:AssociateIpamResourceDiscovery
	ec2:AssociateNatGatewayAddress
	ec2:AssociateRouteServer
	ec2:AssociateRouteTable
	ec2:AssociateSecurityGroupVpc
	ec2:AssociateSubnetCidrBlock
	ec2:AssociateTransitGatewayMulticastDomain
	ec2:AssociateTransitGatewayPolicyTable
	ec2:AssociateTransitGatewayRouteTable
	ec2:AssociateTrunkInterface
	ec2:AssociateVpcCidrBlock
	ec2:AttachClassicLinkVpc
	ec2:AttachImageWatermark
	ec2:AttachInternetGateway
	ec2:AttachNetworkInterface
	ec2:AttachVerifiedAccessTrustProvider
	ec2:AttachVolume

Service prefix	Actions
	ec2:AttachVpnGateway
	ec2:AuthorizeClientVpnIngress
	ec2:AuthorizeSecurityGroupEgress
	ec2:AuthorizeSecurityGroupIngress
	ec2:BundleInstance
	ec2:CancelBundleTask
	ec2:CancelCapacityReservation
	ec2:CancelCapacityReservationFleets
	ec2:CancelConversionTask
	ec2:CancelDeclarativePoliciesReport
	ec2:CancelExportTask
	ec2:CancelImageLaunchPermission
	ec2:CancelImportTask
	ec2:CancelReservedInstancesListing
	ec2:CancelSpotFleetRequests
	ec2:CancelSpotInstanceRequests
	ec2:ConfirmProductInstance
	ec2:CopyFpgaImage
	ec2:CopyImage
	ec2:CopySnapshot
	ec2:CopyVolumes

Service prefix	Actions
	ec2:CreateCapacityManagerDataExport
	ec2:CreateCapacityReservation
	ec2:CreateCapacityReservationBySplitting
	ec2:CreateCapacityReservationCancellationQuote
	ec2:CreateCapacityReservationFleet
	ec2:CreateCarrierGateway
	ec2:CreateClientVpnEndpoint
	ec2:CreateClientVpnRoute
	ec2:CreateCoipCidr
	ec2:CreateCoipPool
	ec2:CreateCustomerGateway
	ec2:CreateDefaultSubnet
	ec2:CreateDefaultVpc
	ec2:CreateDelegateMacVolumeOwnershipTask
	ec2:CreateDhcpOptions
	ec2:CreateEgressOnlyInternetGateway
	ec2:CreateFleet
	ec2:CreateFlowLogs
	ec2:CreateFpgaImage
	ec2:CreateImage
	ec2:CreateImageUsageReport

Service prefix	Actions
	ec2:CreateInstanceConnectEndpoint
	ec2:CreateInstanceEventWindow
	ec2:CreateInstanceExportTask
	ec2:CreateInternetGateway
	ec2:CreateInterruptibleCapacityReservationAllocation
	ec2:CreateIpam
	ec2:CreateIpamExternalResourceVerificationToken
	ec2:CreateIpamPolicy
	ec2:CreateIpamPool
	ec2:CreateIpamPrefixListResolver
	ec2:CreateIpamPrefixListResolverTarget
	ec2:CreateIpamResourceDiscovery
	ec2:CreateIpamScope
	ec2:CreateKeyPair
	ec2:CreateLaunchTemplateVersion
	ec2:CreateLocalGatewayRoute
	ec2:CreateLocalGatewayRouteTable
	ec2:CreateLocalGatewayRouteTableVirtualInterfaceGroupAssociation
	ec2:CreateLocalGatewayRouteTableVpcAssociation
	ec2:CreateLocalGatewayVirtualInterface

Service prefix	Actions
	ec2:CreateLocalGatewayVirtualInterfaceGroup
	ec2:CreateMacSystemIntegrityProtectionModificationTask
	ec2:CreateManagedPrefixList
	ec2:CreateNatGateway
	ec2:CreateNetworkAcl
	ec2:CreateNetworkAclEntry
	ec2:CreateNetworkInsightsAccessScope
	ec2:CreateNetworkInsightsPath
	ec2:CreateNetworkInterface
	ec2:CreateNetworkInterfacePermission
	ec2:CreatePlacementGroup
	ec2:CreatePublicIpv4Pool
	ec2:CreateReplaceRootVolumeTask
	ec2:CreateReservedInstancesListing
	ec2:CreateRestoreImageTask
	ec2:CreateRoute
	ec2:CreateRouteServer
	ec2:CreateRouteServerEndpoint
	ec2:CreateRouteServerPeer
	ec2:CreateRouteTable
	ec2:CreateSecondaryNetwork

Service prefix	Actions
	ec2:CreateSecondarySubnet
	ec2:CreateSecurityGroup
	ec2:CreateSnapshots
	ec2:CreateSpotDatafeedSubscription
	ec2:CreateStoreImageTask
	ec2:CreateSubnet
	ec2:CreateSubnetCidrReservation
	ec2:CreateTrafficMirrorFilter
	ec2:CreateTrafficMirrorFilterRule
	ec2:CreateTrafficMirrorSession
	ec2:CreateTrafficMirrorTarget
	ec2:CreateTransitGateway
	ec2:CreateTransitGatewayConnect
	ec2:CreateTransitGatewayConnectPeer
	ec2:CreateTransitGatewayMeteringPolicy
	ec2:CreateTransitGatewayMeteringPolicyEntry
	ec2:CreateTransitGatewayMulticastDomain
	ec2:CreateTransitGatewayPeeringAttachment
	ec2:CreateTransitGatewayPolicyTable
	ec2:CreateTransitGatewayPrefixListReference
	ec2:CreateTransitGatewayRoute

Service prefix	Actions
	ec2:CreateTransitGatewayRouteTable
	ec2:CreateTransitGatewayRouteTableAnnouncement
	ec2:CreateTransitGatewayVpcAttachment
	ec2:CreateVerifiedAccessEndpoint
	ec2:CreateVerifiedAccessGroup
	ec2:CreateVerifiedAccessInstance
	ec2:CreateVerifiedAccessTrustProvider
	ec2:CreateVolume
	ec2:CreateVpc
	ec2:CreateVpcBlockPublicAccessExclusion
	ec2:CreateVpcEncryptionControl
	ec2:CreateVpcEndpoint
	ec2:CreateVpcEndpointConnectionNotification
	ec2:CreateVpcEndpointServiceConfiguration
	ec2:CreateVpcPeeringConnection
	ec2:CreateVpnConcentrator
	ec2:CreateVpnConnection
	ec2:CreateVpnConnectionRoute
	ec2:CreateVpnGateway
	ec2:DeleteCapacityManagerDataExport
	ec2:DeleteCarrierGateway

Service prefix	Actions
	ec2:DeleteClientVpnEndpoint
	ec2:DeleteClientVpnRoute
	ec2:DeleteCoipCidr
	ec2:DeleteCoipPool
	ec2:DeleteCustomerGateway
	ec2:DeleteDhcpOptions
	ec2:DeleteEgressOnlyInternetGateway
	ec2:DeleteFleets
	ec2:DeleteFlowLogs
	ec2:DeleteFpgaImage
	ec2:DeleteImageUsageReport
	ec2:DeleteInstanceConnectEndpoint
	ec2:DeleteInstanceEventWindow
	ec2:DeleteInternetGateway
	ec2:DeleteIpam
	ec2:DeleteIpamExternalResourceVerificationToken
	ec2:DeleteIpamPolicy
	ec2:DeleteIpamPool
	ec2:DeleteIpamPrefixListResolver
	ec2:DeleteIpamPrefixListResolverTarget
	ec2:DeleteIpamResourceDiscovery

Service prefix	Actions
	ec2:DeleteIpamScope
	ec2:DeleteKeyPair
	ec2:DeleteLaunchTemplate
	ec2:DeleteLaunchTemplateVersions
	ec2:DeleteLocalGatewayRoute
	ec2:DeleteLocalGatewayRouteTable
	ec2:DeleteLocalGatewayRouteTableVirtualInterfaceGroupAssociation
	ec2:DeleteLocalGatewayRouteTableVpcAssociation
	ec2:DeleteLocalGatewayVirtualInterface
	ec2:DeleteLocalGatewayVirtualInterfaceGroup
	ec2:DeleteManagedPrefixList
	ec2:DeleteNatGateway
	ec2:DeleteNetworkAcl
	ec2:DeleteNetworkAclEntry
	ec2:DeleteNetworkInsightsAccessScope
	ec2:DeleteNetworkInsightsAccessScopeAnalysis
	ec2:DeleteNetworkInsightsAnalysis
	ec2:DeleteNetworkInsightsPath
	ec2:DeleteNetworkInterface
	ec2:DeleteNetworkInterfacePermission

Service prefix	Actions
	ec2:DeletePlacementGroup
	ec2:DeletePublicIpv4Pool
	ec2:DeleteQueuedReservedInstances
	ec2:DeleteRoute
	ec2:DeleteRouteServer
	ec2:DeleteRouteServerEndpoint
	ec2:DeleteRouteServerPeer
	ec2:DeleteRouteTable
	ec2:DeleteSecondaryNetwork
	ec2:DeleteSecondarySubnet
	ec2:DeleteSecurityGroup
	ec2:DeleteSpotDatafeedSubscription
	ec2:DeleteSubnet
	ec2:DeleteSubnetCidrReservation
	ec2:DeleteTrafficMirrorFilter
	ec2:DeleteTrafficMirrorFilterRule
	ec2:DeleteTrafficMirrorSession
	ec2:DeleteTrafficMirrorTarget
	ec2:DeleteTransitGateway
	ec2:DeleteTransitGatewayClientVpnAttachment
	ec2:DeleteTransitGatewayConnect

Service prefix	Actions
	ec2:DeleteTransitGatewayConnectPeer
	ec2:DeleteTransitGatewayMeteringPolicy
	ec2:DeleteTransitGatewayMeteringPolicyEntry
	ec2:DeleteTransitGatewayMulticastDomain
	ec2:DeleteTransitGatewayPeeringAttachment
	ec2:DeleteTransitGatewayPolicyTable
	ec2:DeleteTransitGatewayPrefixListReference
	ec2:DeleteTransitGatewayRoute
	ec2:DeleteTransitGatewayRouteTable
	ec2:DeleteTransitGatewayRouteTableAnnouncement
	ec2:DeleteTransitGatewayVpcAttachment
	ec2:DeleteVerifiedAccessEndpoint
	ec2:DeleteVerifiedAccessGroup
	ec2:DeleteVerifiedAccessInstance
	ec2:DeleteVerifiedAccessTrustProvider
	ec2:DeleteVolume
	ec2:DeleteVpc
	ec2:DeleteVpcBlockPublicAccessExclusion
	ec2:DeleteVpcEncryptionControl
	ec2:DeleteVpcEndpointConnectionNotifications
	ec2:DeleteVpcEndpointServiceConfigurations

Service prefix	Actions
	ec2:DeleteVpcEndpoints
	ec2:DeleteVpcPeeringConnection
	ec2:DeleteVpnConcentrator
	ec2:DeleteVpnConnection
	ec2:DeleteVpnConnectionRoute
	ec2:DeleteVpnGateway
	ec2:DeprovisionByoipCidr
	ec2:DeprovisionIpamByoasn
	ec2:DeprovisionIpamPoolCidr
	ec2:DeprovisionPublicIpv4PoolCidr
	ec2:DeregisterImage
	ec2:DeregisterInstanceEventNotificationAttributes
	ec2:DeregisterTransitGatewayMulticastGroupMembers
	ec2:DeregisterTransitGatewayMulticastGroupSources
	ec2:DescribeAccountAttributes
	ec2:DescribeAddressTransfers
	ec2:DescribeAddresses
	ec2:DescribeAddressesAttribute
	ec2:DescribeAggregateIdFormat
	ec2:DescribeAvailabilityZones
	ec2:DescribeAwsNetworkPerformanceMetricSubscriptions

Service prefix	Actions
	ec2:DescribeBundleTasks
	ec2:DescribeByoipCidrs
	ec2:DescribeCapacityBlockExtensionHistory
	ec2:DescribeCapacityBlockExtensionOfferings
	ec2:DescribeCapacityBlockStatus
	ec2:DescribeCapacityBlocks
	ec2:DescribeCapacityManagerDataExports
	ec2:DescribeCapacityReservationBillingRequests
	ec2:DescribeCapacityReservationCancellationQuotes
	ec2:DescribeCapacityReservationFleets
	ec2:DescribeCapacityReservationTopology
	ec2:DescribeCapacityReservations
	ec2:DescribeCarrierGateways
	ec2:DescribeClassicLinkInstances
	ec2:DescribeClientVpnAuthorizationRules
	ec2:DescribeClientVpnConnections
	ec2:DescribeClientVpnEndpoints
	ec2:DescribeClientVpnRoutes
	ec2:DescribeClientVpnTargetNetworks
	ec2:DescribeCoipPools
	ec2:DescribeConversionTasks

Service prefix	Actions
	ec2:DescribeCustomerGateways
	ec2:DescribeDeclarativePoliciesReports
	ec2:DescribeDhcpOptions
	ec2:DescribeEgressOnlyInternetGateways
	ec2:DescribeElasticGpus
	ec2:DescribeExportImageTasks
	ec2:DescribeExportTasks
	ec2:DescribeFastLaunchImages
	ec2:DescribeFastSnapshotRestores
	ec2:DescribeFleetHistory
	ec2:DescribeFleetInstances
	ec2:DescribeFleets
	ec2:DescribeFlowLogs
	ec2:DescribeFpgaImageAttribute
	ec2:DescribeFpgaImages
	ec2:DescribeHostReservationOfferings
	ec2:DescribeHostReservations
	ec2:DescribeHosts
	ec2:DescribeIamInstanceProfileAssociations
	ec2:DescribeIdFormat
	ec2:DescribeIdentityIdFormat

Service prefix	Actions
	ec2:DescribeImageAttribute
	ec2:DescribeImageReferences
	ec2:DescribeImageUsageReportEntries
	ec2:DescribeImageUsageReports
	ec2:DescribeImportImageTasks
	ec2:DescribeImportSnapshotTasks
	ec2:DescribeInstanceConnectEndpoints
	ec2:DescribeInstanceCreditSpecifications
	ec2:DescribeInstanceEventNotificationAttributes
	ec2:DescribeInstanceEventWindows
	ec2:DescribeInstanceImageMetadata
	ec2:DescribeInstanceSqlHaHistoryStates
	ec2:DescribeInstanceSqlHaStates
	ec2:DescribeInstanceTopology
	ec2:DescribeInstanceTypes
	ec2:DescribeInternetGateways
	ec2:DescribeIpamByoasn
	ec2:DescribeIpamExternalResourceVerificationTokens
	ec2:DescribeIpamPolicies
	ec2:DescribeIpamPoolAllocations
	ec2:DescribeIpamPools

Service prefix	Actions
	ec2:DescribeIpamPrefixListResolverTargets
	ec2:DescribeIpamPrefixListResolvers
	ec2:DescribeIpamResourceDiscoveries
	ec2:DescribeIpamResourceDiscoveryAssociations
	ec2:DescribeIpamScopes
	ec2:DescribeIpams
	ec2:DescribeIpv6Pools
	ec2:DescribeKeyPairs
	ec2:DescribeLocalGatewayRouteTableVirtualInterfaceGroupAssociations
	ec2:DescribeLocalGatewayRouteTableVpcAssociations
	ec2:DescribeLocalGatewayRouteTables
	ec2:DescribeLocalGatewayVirtualInterfaceGroups
	ec2:DescribeLocalGatewayVirtualInterfaces
	ec2:DescribeLocalGateways
	ec2:DescribeLockedSnapshots
	ec2:DescribeMacHosts
	ec2:DescribeMacModificationTasks
	ec2:DescribeManagedPrefixLists
	ec2:DescribeMovingAddresses
	ec2:DescribeNatGateways

Service prefix	Actions
	ec2:DescribeNetworkAcls
	ec2:DescribeNetworkInsightsAccessScopeAnalyses
	ec2:DescribeNetworkInsightsAccessScopes
	ec2:DescribeNetworkInsightsAnalyses
	ec2:DescribeNetworkInsightsPaths
	ec2:DescribeNetworkInterfaceAttribute
	ec2:DescribeNetworkInterfacePermissions
	ec2:DescribeNetworkInterfaces
	ec2:DescribeOutpostLags
	ec2:DescribePlacementGroups
	ec2:DescribePrefixLists
	ec2:DescribePrincipalIdFormat
	ec2:DescribePublicIpv4Pools
	ec2:DescribeRegions
	ec2:DescribeReplaceRootVolumeTasks
	ec2:DescribeReservedInstances
	ec2:DescribeReservedInstancesListings
	ec2:DescribeReservedInstancesModifications
	ec2:DescribeReservedInstancesOfferings
	ec2:DescribeRouteServerEndpoints
	ec2:DescribeRouteServerPeers

Service prefix	Actions
	ec2:DescribeRouteServers
	ec2:DescribeRouteTables
	ec2:DescribeScheduledInstanceAvailability
	ec2:DescribeScheduledInstances
	ec2:DescribeSecondaryInterfaces
	ec2:DescribeSecondaryNetworks
	ec2:DescribeSecondarySubnets
	ec2:DescribeSecurityGroupReferences
	ec2:DescribeSecurityGroupRules
	ec2:DescribeSecurityGroupVpcAssociations
	ec2:DescribeSecurityGroups
	ec2:DescribeServiceLinkVirtualInterfaces
	ec2:DescribeSnapshotAttribute
	ec2:DescribeSnapshotTierStatus
	ec2:DescribeSpotDatafeedSubscription
	ec2:DescribeSpotFleetInstances
	ec2:DescribeSpotFleetRequestHistory
	ec2:DescribeSpotFleetRequests
	ec2:DescribeSpotInstanceRequests
	ec2:DescribeSpotPriceHistory
	ec2:DescribeStaleSecurityGroups

Service prefix	Actions
	ec2:DescribeStoreImageTasks
	ec2:DescribeTrafficMirrorFilterRules
	ec2:DescribeTrafficMirrorFilters
	ec2:DescribeTrafficMirrorSessions
	ec2:DescribeTrafficMirrorTargets
	ec2:DescribeTransitGatewayAttachments
	ec2:DescribeTransitGatewayConnectPeers
	ec2:DescribeTransitGatewayConnects
	ec2:DescribeTransitGatewayMeteringPolicies
	ec2:DescribeTransitGatewayMulticastDomains
	ec2:DescribeTransitGatewayPeeringAttachments
	ec2:DescribeTransitGatewayPolicyTables
	ec2:DescribeTransitGatewayRouteTableAnnouncements
	ec2:DescribeTransitGatewayRouteTables
	ec2:DescribeTransitGatewayVpcAttachments
	ec2:DescribeTransitGateways
	ec2:DescribeTrunkInterfaceAssociations
	ec2:DescribeVerifiedAccessEndpoints
	ec2:DescribeVerifiedAccessGroups
	ec2:DescribeVerifiedAccessInstanceLoggingConfigurations
	ec2:DescribeVerifiedAccessInstances

Service prefix	Actions
	ec2:DescribeVerifiedAccessTrustProviders
	ec2:DescribeVolumeAttribute
	ec2:DescribeVolumeStatus
	ec2:DescribeVolumes
	ec2:DescribeVolumesModifications
	ec2:DescribeVpcAttribute
	ec2:DescribeVpcBlockPublicAccessExclusions
	ec2:DescribeVpcBlockPublicAccessOptions
	ec2:DescribeVpcClassicLink
	ec2:DescribeVpcClassicLinkDnsSupport
	ec2:DescribeVpcEncryptionControls
	ec2:DescribeVpcEndpointAssociations
	ec2:DescribeVpcEndpointConnectionNotifications
	ec2:DescribeVpcEndpointConnections
	ec2:DescribeVpcEndpointServiceConfigurations
	ec2:DescribeVpcEndpointServicePermissions
	ec2:DescribeVpcEndpointServices
	ec2:DescribeVpcEndpoints
	ec2:DescribeVpcPeeringConnections
	ec2:DescribeVpcs
	ec2:DescribeVpnConcentrators

Service prefix	Actions
	ec2:DescribeVpnConnections
	ec2:DescribeVpnGateways
	ec2:DetachClassicLinkVpc
	ec2:DetachImageWatermark
	ec2:DetachInternetGateway
	ec2:DetachNetworkInterface
	ec2:DetachVerifiedAccessTrustProvider
	ec2:DetachVolume
	ec2:DetachVpnGateway
	ec2:DisableAddressTransfer
	ec2:DisableAllowedImagesSettings
	ec2:DisableAwsNetworkPerformanceMetricSubscription
	ec2:DisableCapacityManager
	ec2:DisableEbsEncryptionByDefault
	ec2:DisableFastLaunch
	ec2:DisableFastSnapshotRestores
	ec2:DisableImage
	ec2:DisableImageBlockPublicAccess
	ec2:DisableImageDeprecation
	ec2:DisableImageDeregistrationProtection
	ec2:DisableInstanceSqlHaStandbyDetections

Service prefix	Actions
	ec2:DisableIpamOrganizationAdminAccount
	ec2:DisableIpamPolicy
	ec2:DisableRouteServerPropagation
	ec2:DisableSerialConsoleAccess
	ec2:DisableSnapshotBlockPublicAccess
	ec2:DisableTransitGatewayRouteTablePropagation
	ec2:DisableVgwRoutePropagation
	ec2:DisableVpcClassicLink
	ec2:DisableVpcClassicLinkDnsSupport
	ec2:DisassociateAddress
	ec2:DisassociateCapacityReservationBillingOwner
	ec2:DisassociateClientVpnTargetNetwork
	ec2:DisassociateEnclaveCertificateIamRole
	ec2:DisassociateIamInstanceProfile
	ec2:DisassociateInstanceEventWindow
	ec2:DisassociateIpamByoasn
	ec2:DisassociateIpamResourceDiscovery
	ec2:DisassociateNatGatewayAddress
	ec2:DisassociateRouteServer
	ec2:DisassociateRouteTable
	ec2:DisassociateSecurityGroupVpc

Service prefix	Actions
	ec2:DisassociateSubnetCidrBlock
	ec2:DisassociateTransitGatewayMulticastDomain
	ec2:DisassociateTransitGatewayPolicyTable
	ec2:DisassociateTransitGatewayRouteTable
	ec2:DisassociateTrunkInterface
	ec2:DisassociateVpcCidrBlock
	ec2:EnableAddressTransfer
	ec2:EnableAllowedImagesSettings
	ec2:EnableAwsNetworkPerformanceMetricSubscription
	ec2:EnableCapacityManager
	ec2:EnableEbsEncryptionByDefault
	ec2:EnableFastLaunch
	ec2:EnableFastSnapshotRestores
	ec2:EnableImage
	ec2:EnableImageBlockPublicAccess
	ec2:EnableImageDeprecation
	ec2:EnableImageDeregistrationProtection
	ec2:EnableInstanceSqlHaStandbyDetections
	ec2:EnableIpamOrganizationAdminAccount
	ec2:EnableIpamPolicy
	ec2:EnableReachabilityAnalyzerOrganizationSharing

Service prefix	Actions
	ec2:EnableRouteServerPropagation
	ec2:EnableSerialConsoleAccess
	ec2:EnableSnapshotBlockPublicAccess
	ec2:EnableTransitGatewayRouteTablePropagation
	ec2:EnableVgwRoutePropagation
	ec2:EnableVolumeIO
	ec2:EnableVpcClassicLink
	ec2:EnableVpcClassicLinkDnsSupport
	ec2:ExportClientVpnClientCertificateRevocationList
	ec2:ExportClientVpnClientConfiguration
	ec2:ExportImage
	ec2:ExportTransitGatewayRoutes
	ec2:ExportVerifiedAccessInstanceClientConfiguration
	ec2:GetActiveVpnTunnelStatus
	ec2:GetAllowedImagesSettings
	ec2:GetAssociatedEnclaveCertificateIamRoles
	ec2:GetAssociatedIpv6PoolCidrs
	ec2:GetAwsNetworkPerformanceData
	ec2:GetCapacityManagerAttributes
	ec2:GetCapacityManagerMetricData
	ec2:GetCapacityManagerMetricDimensions

Service prefix	Actions
	ec2:GetCapacityReservationUsage
	ec2:GetCoipPoolUsage
	ec2:GetConsoleOutput
	ec2:GetConsoleScreenshot
	ec2:GetDeclarativePoliciesReportSummary
	ec2:GetDefaultCreditSpecification
	ec2:GetEbsDefaultKmsKeyId
	ec2:GetEbsEncryptionByDefault
	ec2:GetEnabledIpamPolicy
	ec2:GetFlowLogsIntegrationTemplate
	ec2:GetGroupsForCapacityReservation
	ec2:GetHostReservationPurchasePreview
	ec2:GetImageAncestry
	ec2:GetImageBlockPublicAccessState
	ec2:GetInstanceMetadataDefaults
	ec2:GetInstanceTpmEkPub
	ec2:GetInstanceTypesFromInstanceRequirements
	ec2:GetInstanceUefiData
	ec2:GetIpamAddressHistory
	ec2:GetIpamDiscoveredAccounts
	ec2:GetIpamDiscoveredPublicAddresses

Service prefix	Actions
	ec2:GetIpamDiscoveredResourceCidrs
	ec2:GetIpamPolicyAllocationRules
	ec2:GetIpamPolicyOrganizationTargets
	ec2:GetIpamPoolAllocations
	ec2:GetIpamPoolCidrs
	ec2:GetIpamPrefixListResolverRules
	ec2:GetIpamPrefixListResolverVersionEntries
	ec2:GetIpamPrefixListResolverVersions
	ec2:GetIpamResourceCidrs
	ec2:GetLaunchTemplateData
	ec2:GetManagedPrefixListAssociations
	ec2:GetManagedPrefixListEntries
	ec2:GetManagedResourceVisibility
	ec2:GetNetworkInsightsAccessScopeAnalysisFindings
	ec2:GetNetworkInsightsAccessScopeContent
	ec2:GetPasswordData
	ec2:GetReservedInstancesExchangeQuote
	ec2:GetRouteServerAssociations
	ec2:GetRouteServerPropagations
	ec2:GetRouteServerRoutingDatabase
	ec2:GetSecurityGroupsForVpc

Service prefix	Actions
	ec2:GetSerialConsoleAccessStatus
	ec2:GetSnapshotBlockPublicAccessState
	ec2:GetSpotPlacementScores
	ec2:GetSubnetCidrReservations
	ec2:GetTransitGatewayAttachmentPropagations
	ec2:GetTransitGatewayMeteringPolicyEntries
	ec2:GetTransitGatewayMulticastDomainAssociations
	ec2:GetTransitGatewayPolicyTableAssociations
	ec2:GetTransitGatewayPolicyTableEntries
	ec2:GetTransitGatewayPrefixListReferences
	ec2:GetTransitGatewayRouteTableAssociations
	ec2:GetTransitGatewayRouteTablePropagations
	ec2:GetVerifiedAccessEndpointPolicy
	ec2:GetVerifiedAccessEndpointTargets
	ec2:GetVerifiedAccessGroupPolicy
	ec2:GetVpcResourcesBlockingEncryptionEnforcement
	ec2:GetVpnConnectionDeviceSampleConfiguration
	ec2:GetVpnConnectionDeviceTypes
	ec2:GetVpnTunnelReplacementStatus
	ec2:ImportClientVpnClientCertificateRevocationList
	ec2:ImportImage

Service prefix	Actions
	ec2:ImportInstance
	ec2:ImportKeyPair
	ec2:ImportSnapshot
	ec2:ImportVolume
	ec2:ListImagesInRecycleBin
	ec2:ListSnapshotsInRecycleBin
	ec2:ListVolumesInRecycleBin
	ec2:LockSnapshot
	ec2:ModifyAddressAttribute
	ec2:ModifyAvailabilityZoneGroup
	ec2:ModifyCapacityReservation
	ec2:ModifyCapacityReservationFleet
	ec2:ModifyClientVpnEndpoint
	ec2:ModifyDefaultCreditSpecification
	ec2:ModifyEbsDefaultKmsKeyId
	ec2:ModifyFleet
	ec2:ModifyFpgaImageAttribute
	ec2:ModifyHosts
	ec2:ModifyIdFormat
	ec2:ModifyIdentityIdFormat
	ec2:ModifyImageAttribute

Service prefix	Actions
	ec2:ModifyInstanceAttribute
	ec2:ModifyInstanceCapacityReservationAttributes
	ec2:ModifyInstanceConnectEndpoint
	ec2:ModifyInstanceCpuOptions
	ec2:ModifyInstanceCreditSpecification
	ec2:ModifyInstanceEventStartTime
	ec2:ModifyInstanceEventWindow
	ec2:ModifyInstanceMaintenanceOptions
	ec2:ModifyInstanceMetadataDefaults
	ec2:ModifyInstanceMetadataOptions
	ec2:ModifyInstanceNetworkPerformanceOptions
	ec2:ModifyInstancePlacement
	ec2:ModifyIpam
	ec2:ModifyIpamPolicyAllocationRules
	ec2:ModifyIpamPool
	ec2:ModifyIpamPoolAllocation
	ec2:ModifyIpamPrefixListResolver
	ec2:ModifyIpamPrefixListResolverTarget
	ec2:ModifyIpamResourceCidr
	ec2:ModifyIpamResourceDiscovery
	ec2:ModifyIpamScope

Service prefix	Actions
	ec2:ModifyLaunchTemplate
	ec2:ModifyLocalGatewayRoute
	ec2:ModifyManagedPrefixList
	ec2:ModifyManagedResourceVisibility
	ec2:ModifyNetworkInterfaceAttribute
	ec2:ModifyPrivateDnsNameOptions
	ec2:ModifyPublicIpDnsNameOptions
	ec2:ModifyReservedInstances
	ec2:ModifyRouteServer
	ec2:ModifySecurityGroupRules
	ec2:ModifySnapshotAttribute
	ec2:ModifySnapshotTier
	ec2:ModifySpotFleetRequest
	ec2:ModifySubnetAttribute
	ec2:ModifyTrafficMirrorFilterNetworkServices
	ec2:ModifyTrafficMirrorFilterRule
	ec2:ModifyTrafficMirrorSession
	ec2:ModifyTransitGateway
	ec2:ModifyTransitGatewayMeteringPolicy
	ec2:ModifyTransitGatewayPrefixListReference
	ec2:ModifyTransitGatewayVpcAttachment

Service prefix	Actions
	ec2:ModifyVerifiedAccessEndpoint
	ec2:ModifyVerifiedAccessEndpointPolicy
	ec2:ModifyVerifiedAccessGroup
	ec2:ModifyVerifiedAccessGroupPolicy
	ec2:ModifyVerifiedAccessInstance
	ec2:ModifyVerifiedAccessInstanceLoggingConfiguration
	ec2:ModifyVerifiedAccessTrustProvider
	ec2:ModifyVolume
	ec2:ModifyVolumeAttribute
	ec2:ModifyVpcAttribute
	ec2:ModifyVpcBlockPublicAccessExclusion
	ec2:ModifyVpcBlockPublicAccessOptions
	ec2:ModifyVpcEncryptionControl
	ec2:ModifyVpcEndpoint
	ec2:ModifyVpcEndpointConnectionNotification
	ec2:ModifyVpcEndpointServiceConfiguration
	ec2:ModifyVpcEndpointServicePayerResponsibility
	ec2:ModifyVpcEndpointServicePermissions
	ec2:ModifyVpcPeeringConnectionOptions
	ec2:ModifyVpcTenancy
	ec2:ModifyVpnConnection

Service prefix	Actions
	ec2:ModifyVpnConnectionOptions
	ec2:ModifyVpnTunnelCertificate
	ec2:ModifyVpnTunnelOptions
	ec2:MonitorInstances
	ec2:MoveAddressToVpc
	ec2:MoveByoipCidrToIpam
	ec2:MoveCapacityReservationInstances
	ec2:ProvisionByoipCidr
	ec2:ProvisionIpamByoasn
	ec2:ProvisionIpamPoolCidr
	ec2:ProvisionPublicIpv4PoolCidr
	ec2:PurchaseCapacityBlockExtension
	ec2:PurchaseHostReservation
	ec2:PurchaseReservedInstancesOffering
	ec2:PurchaseScheduledInstances
	ec2:RebootInstances
	ec2:RegisterImage
	ec2:RegisterInstanceEventNotificationAttributes
	ec2:RegisterTransitGatewayMulticastGroupMembers
	ec2:RegisterTransitGatewayMulticastGroupSources
	ec2:RejectCapacityReservationBillingOwnership

Service prefix	Actions
	ec2:RejectTransitGatewayClientVpnAttachment
	ec2:RejectTransitGatewayMulticastDomainAssociations
	ec2:RejectTransitGatewayPeeringAttachment
	ec2:RejectTransitGatewayVpcAttachment
	ec2:RejectVpcEndpointConnections
	ec2:RejectVpcPeeringConnection
	ec2:ReleaseAddress
	ec2:ReleaseHosts
	ec2:ReleaseIpamPoolAllocation
	ec2:ReplaceIamInstanceProfileAssociation
	ec2:ReplaceImageCriteriaInAllowedImagesSettings
	ec2:ReplaceNetworkAclAssociation
	ec2:ReplaceNetworkAclEntry
	ec2:ReplaceRoute
	ec2:ReplaceRouteTableAssociation
	ec2:ReplaceTransitGatewayRoute
	ec2:ReplaceVpnTunnel
	ec2:ReportInstanceStatus
	ec2:RequestSpotFleet
	ec2:RequestSpotInstances
	ec2:ResetAddressAttribute

Service prefix	Actions
	ec2:ResetEbsDefaultKmsKeyId
	ec2:ResetFpgaImageAttribute
	ec2:ResetImageAttribute
	ec2:ResetInstanceAttribute
	ec2:ResetNetworkInterfaceAttribute
	ec2:ResetSnapshotAttribute
	ec2:RestoreAddressToClassic
	ec2:RestoreImageFromRecycleBin
	ec2:RestoreManagedPrefixListVersion
	ec2:RestoreSnapshotFromRecycleBin
	ec2:RestoreSnapshotTier
	ec2:RestoreVolumeFromRecycleBin
	ec2:RevokeClientVpnIngress
	ec2:RevokeSecurityGroupEgress
	ec2:RevokeSecurityGroupIngress
	ec2:RunInstances
	ec2:RunScheduledInstances
	ec2:SearchLocalGatewayRoutes
	ec2:SearchTransitGatewayMulticastGroups
	ec2:SearchTransitGatewayRoutes
	ec2:SendDiagnosticInterrupt

Service prefix	Actions
	ec2:StartDeclarativePoliciesReport
	ec2:StartInstances
	ec2:StartNetworkInsightsAccessScopeAnalysis
	ec2:StartNetworkInsightsAnalysis
	ec2:StartVpcEndpointServicePrivateDnsVerification
	ec2:TerminateClientVpnConnections
	ec2:UnassignIpv6Addresses
	ec2:UnassignPrivateIpAddresses
	ec2:UnassignPrivateNatGatewayAddress
	ec2:UnlockSnapshot
	ec2:UnmonitorInstances
	ec2:UpdateCapacityManagerOrganizationsAccess
	ec2:UpdateInterruptibleCapacityReservationAllocation
	ec2:UpdateSecurityGroupRuleDescriptionsEgress
	ec2:UpdateSecurityGroupRuleDescriptionsIngress
	ec2:WithdrawByoipCidr

Service prefix	Actions
ecr	ecr:BatchCheckLayerAvailability ecr:BatchDeleteImage ecr:BatchGetImage ecr:BatchGetRepositoryScanningConfiguration ecr:CompleteLayerUpload ecr:CreatePullThroughCacheRule ecr:CreateRepositoryCreationTemplate ecr:DeleteLifecyclePolicy ecr:DeletePullThroughCacheRule ecr:DeleteRegistryPolicy ecr:DeleteRepository ecr:DeleteRepositoryCreationTemplate ecr:DeleteRepositoryPolicy ecr:DeleteSigningConfiguration ecr:DescribeImageReplicationStatus ecr:DescribeImageScanFindings ecr:DescribeImages ecr:DescribePullThroughCacheRules ecr:DescribeRegistry ecr:DescribeRepositories ecr:DescribeRepositoryCreationTemplates

Service prefix	Actions
	ecr:GetAccountSetting ecr:GetAuthorizationToken ecr:GetDownloadUrlForLayer ecr:GetLifecyclePolicy ecr:GetLifecyclePolicyPreview ecr:GetRegistryPolicy ecr:GetRegistryScanningConfiguration ecr:GetRepositoryPolicy ecr:GetSigningConfiguration ecr:InitiateLayerUpload ecr:ListImages ecr:ListPullTimeUpdateExclusions ecr:PutAccountSetting ecr:PutImage ecr:PutImageScanningConfiguration ecr:PutRegistryPolicy ecr:PutRegistryScanningConfiguration ecr:PutReplicationConfiguration ecr:StartImageScan ecr:StartLifecyclePolicyPreview ecr:UpdatePullThroughCacheRule

Service prefix	Actions
	ecr:UpdateRepositoryCreationTemplate ecr:UploadLayerPart ecr:ValidatePullThroughCacheRule

Service prefix	Actions
ecr-public	ecr-public:BatchCheckLayerAvailability
	ecr-public:BatchDeleteImage
	ecr-public:CompleteLayerUpload
	ecr-public:CreateRepository
	ecr-public:DeleteRepository
	ecr-public:DeleteRepositoryPolicy
	ecr-public:DescribeImages
	ecr-public:DescribeRegistries
	ecr-public:DescribeRepositories
	ecr-public:GetAuthorizationToken
	ecr-public:GetRegistryCatalogData
	ecr-public:GetRepositoryCatalogData
	ecr-public:GetRepositoryPolicy
	ecr-public:InitiateLayerUpload
	ecr-public:PutImage
	ecr-public:PutRegistryCatalogData
	ecr-public:PutRepositoryCatalogData
	ecr-public:SetRepositoryPolicy
	ecr-public:UploadLayerPart

Service prefix	Actions
ecs	ecs:ContinueServiceDeployment ecs:CreateCapacityProvider ecs:CreateCluster ecs:CreateDaemon ecs:CreateService ecs:CreateTaskSet ecs:DeleteAccountSetting ecs:DeleteAttributes ecs:DeleteCapacityProvider ecs:DeleteCluster ecs:DeleteDaemon ecs:DeleteDaemonTaskDefinition ecs:DeleteExpressGatewayService ecs:DeleteService ecs:DeleteTaskDefinitions ecs:DeleteTaskSet ecs:DeregisterContainerInstance ecs:DeregisterTaskDefinition ecs:DescribeCapacityProviders ecs:DescribeClusters ecs:DescribeContainerInstances

Service prefix	Actions
	ecs:DescribeDaemon ecs:DescribeDaemonDeployments ecs:DescribeDaemonRevisions ecs:DescribeDaemonTaskDefinition ecs:DescribeExpressGatewayService ecs:DescribeServiceDeployments ecs:DescribeServiceRevisions ecs:DescribeServices ecs:DescribeTaskDefinition ecs:DescribeTaskSets ecs:DescribeTasks ecs:DiscoverPollEndpoint ecs:ExecuteCommand ecs:GetTaskProtection ecs:ListAccountSettings ecs:ListAttributes ecs:ListClusters ecs:ListContainerInstances ecs:ListDaemonDeployments ecs:ListDaemonTaskDefinitions ecs:ListDaemons

Service prefix	Actions
	ecs:ListServiceDeployments ecs:ListServices ecs:ListServicesByNamespace ecs:ListTaskDefinitionFamilies ecs:ListTaskDefinitions ecs:ListTasks ecs:PutAccountSetting ecs:PutAccountSettingDefault ecs:PutAttributes ecs:PutClusterCapacityProviders ecs:RegisterContainerInstance ecs:RegisterDaemonTaskDefinition ecs:RunTask ecs:StartTask ecs:StopServiceDeployment ecs:StopTask ecs:SubmitAttachmentStateChanges ecs:SubmitContainerStateChange ecs:SubmitTaskStateChange ecs:UpdateCapacityProvider ecs:UpdateCluster

Service prefix	Actions
	ecs:UpdateClusterSettings ecs:UpdateContainerAgent ecs:UpdateContainerInstancesState ecs:UpdateDaemon ecs:UpdateExpressGatewayService ecs:UpdateService ecs:UpdateServicePrimaryTaskSet ecs:UpdateTaskProtection ecs:UpdateTaskSet

Service prefix	Actions
eks	eks:AssociateAccessPolicy
	eks:AssociateEncryptionConfig
	eks:AssociateIdentityProviderConfig
	eks:CancelUpdate
	eks:CreateAccessEntry
	eks:CreateAddon
	eks:CreateCluster
	eks:CreateEksAnywhereSubscription
	eks:CreateFargateProfile
	eks:CreateNodegroup
	eks:DeleteAccessEntry
	eks:DeleteAddon
	eks:DeleteCapability
	eks:DeleteCluster
	eks:DeleteEksAnywhereSubscription
	eks:DeleteFargateProfile
	eks:DeleteNodegroup
	eks:DeletePodIdentityAssociation
	eks:DeregisterCluster
	eks:DescribeAccessEntry
	eks:DescribeAddon

Service prefix	Actions
	eks:DescribeAddonConfiguration
	eks:DescribeAddonVersions
	eks:DescribeCapability
	eks:DescribeCluster
	eks:DescribeClusterVersions
	eks:DescribeEksAnywhereSubscription
	eks:DescribeFargateProfile
	eks:DescribeIdentityProviderConfig
	eks:DescribeInsight
	eks:DescribeInsightsRefresh
	eks:DescribeNodegroup
	eks:DescribePodIdentityAssociation
	eks:DescribeUpdate
	eks:DisassociateAccessPolicy
	eks:DisassociateIdentityProviderConfig
	eks:ListAccessEntries
	eks:ListAccessPolicies
	eks:ListAddons
	eks:ListAssociatedAccessPolicies
	eks:ListCapabilities
	eks:ListClusters

Service prefix	Actions
	eks:ListEksAnywhereSubscriptions
	eks:ListFargateProfiles
	eks:ListIdentityProviderConfigs
	eks:ListInsights
	eks:ListNodegroups
	eks:ListPodIdentityAssociations
	eks:ListUpdates
	eks:RegisterCluster
	eks:StartInsightsRefresh
	eks:UpdateAccessEntry
	eks:UpdateAddon
	eks:UpdateCapability
	eks:UpdateClusterConfig
	eks:UpdateClusterVersion
	eks:UpdateEksAnywhereSubscription
	eks:UpdateNodegroupConfig
	eks:UpdateNodegroupVersion
	eks:UpdatePodIdentityAssociation

Service prefix	Actions
elasticache	elasticache:AuthorizeCacheSecurityGroupIngress elasticache:BatchApplyUpdateAction elasticache:BatchStopUpdateAction elasticache:CompleteMigration elasticache:CopyServerlessCacheSnapshot elasticache:CopySnapshot elasticache:CreateCacheCluster elasticache:CreateCacheParameterGroup elasticache:CreateCacheSecurityGroup elasticache:CreateCacheSubnetGroup elasticache:CreateGlobalReplicationGroup elasticache:CreateReplicationGroup elasticache:CreateServerlessCache elasticache:CreateServerlessCacheSnapshot elasticache:CreateSnapshot elasticache:CreateUser elasticache:CreateUserGroup elasticache:DecreaseNodeGroupsInGlobalReplicationGroup elasticache:DecreaseReplicaCount elasticache:DeleteCacheCluster elasticache:DeleteCacheParameterGroup

Service prefix	Actions
	elasticache:DeleteCacheSecurityGroup elasticache:DeleteCacheSubnetGroup elasticache:DeleteGlobalReplicationGroup elasticache:DeleteReplicationGroup elasticache:DeleteServerlessCache elasticache:DeleteServerlessCacheSnapshot elasticache:DeleteSnapshot elasticache:DeleteUser elasticache:DeleteUserGroup elasticache:DescribeCacheClusters elasticache:DescribeCacheEngineVersions elasticache:DescribeCacheParameterGroups elasticache:DescribeCacheParameters elasticache:DescribeCacheSecurityGroups elasticache:DescribeCacheSubnetGroups elasticache:DescribeEngineDefaultParameters elasticache:DescribeEvents elasticache:DescribeGlobalReplicationGroups elasticache:DescribeReplicationGroups elasticache:DescribeReservedCacheNodes elasticache:DescribeReservedCacheNodesOfferings

Service prefix	Actions
	elasticache:DescribeServerlessCacheSnapshots elasticache:DescribeServerlessCaches elasticache:DescribeServiceUpdates elasticache:DescribeSnapshots elasticache:DescribeUpdateActions elasticache:DescribeUserGroups elasticache:DescribeUsers elasticache:DisassociateGlobalReplicationGroup elasticache:ExportServerlessCacheSnapshot elasticache:FailoverGlobalReplicationGroup elasticache:IncreaseNodeGroupsInGlobalReplicationGroup elasticache:IncreaseReplicaCount elasticache:ListAllowedNodeTypeModifications elasticache:ModifyCacheCluster elasticache:ModifyCacheParameterGroup elasticache:ModifyCacheSubnetGroup elasticache:ModifyGlobalReplicationGroup elasticache:ModifyReplicationGroup elasticache:ModifyReplicationGroupShardConfiguration elasticache:ModifyServerlessCache elasticache:ModifyUser

Service prefix	Actions
	elasticache:ModifyUserGroup elasticache:PurchaseReservedCacheNodesOffering elasticache:RebalanceSlotsInGlobalReplicationGroup elasticache:RebootCacheCluster elasticache:ResetCacheParameterGroup elasticache:RevokeCacheSecurityGroupIngress elasticache:StartMigration elasticache:TestFailover elasticache:TestMigration

Service prefix	Actions
elasticbeanstalk	elasticbeanstalk:AbortEnvironmentUpdate
	elasticbeanstalk:ApplyEnvironmentManagedAction
	elasticbeanstalk:AssociateEnvironmentOperationsRole
	elasticbeanstalk:CheckDNSAvailability
	elasticbeanstalk:ComposeEnvironments
	elasticbeanstalk:CreateApplication
	elasticbeanstalk:CreateApplicationVersion
	elasticbeanstalk:CreateConfigurationTemplate
	elasticbeanstalk:CreateEnvironment
	elasticbeanstalk:CreatePlatformVersion
	elasticbeanstalk:CreateStorageLocation
	elasticbeanstalk:DeleteApplication
	elasticbeanstalk:DeleteApplicationVersion
	elasticbeanstalk:DeleteConfigurationTemplate
	elasticbeanstalk:DeleteEnvironmentConfiguration
	elasticbeanstalk:DeletePlatformVersion
	elasticbeanstalk:DescribeAccountAttributes
	elasticbeanstalk:DescribeApplicationVersions
	elasticbeanstalk:DescribeApplications
	elasticbeanstalk:DescribeConfigurationOptions
	elasticbeanstalk:DescribeConfigurationSettings

Service prefix	Actions
	elasticbeanstalk:DescribeEnvironmentHealth
	elasticbeanstalk:DescribeEnvironmentManagedActionHistory
	elasticbeanstalk:DescribeEnvironmentManagedActions
	elasticbeanstalk:DescribeEnvironmentResources
	elasticbeanstalk:DescribeEnvironments
	elasticbeanstalk:DescribeEvents
	elasticbeanstalk:DescribeInstancesHealth
	elasticbeanstalk:DescribePlatformVersion
	elasticbeanstalk:DisassociateEnvironmentOperationsRole
	elasticbeanstalk:ListAvailableSolutionStacks
	elasticbeanstalk:ListPlatformBranches
	elasticbeanstalk:ListPlatformVersions
	elasticbeanstalk:RebuildEnvironment
	elasticbeanstalk:RequestEnvironmentInfo
	elasticbeanstalk:RestartAppServer
	elasticbeanstalk:RetrieveEnvironmentInfo
	elasticbeanstalk:SwapEnvironmentCNAMEs
	elasticbeanstalk:TerminateEnvironment
	elasticbeanstalk:UpdateApplication
	elasticbeanstalk:UpdateApplicationResourceLifecycle
	elasticbeanstalk:UpdateApplicationVersion

Service prefix	Actions
	elasticbeanstalk:UpdateConfigurationTemplate elasticbeanstalk:UpdateEnvironment elasticbeanstalk:ValidateConfigurationSettings

Service prefix	Actions
elasticfilesystem	elasticfilesystem:CreateAccessPoint elasticfilesystem:CreateFileSystem elasticfilesystem:CreateMountTarget elasticfilesystem:CreateReplicationConfiguration elasticfilesystem:DeleteAccessPoint elasticfilesystem:DeleteFileSystem elasticfilesystem:DeleteFileSystemPolicy elasticfilesystem:DeleteMountTarget elasticfilesystem:DeleteReplicationConfiguration elasticfilesystem:DescribeAccessPoints elasticfilesystem:DescribeAccountPreferences elasticfilesystem:DescribeBackupPolicy elasticfilesystem:DescribeFileSystemPolicy elasticfilesystem:DescribeFileSystems elasticfilesystem:DescribeLifecycleConfiguration elasticfilesystem:DescribeMountTargetSecurityGroups elasticfilesystem:DescribeMountTargets elasticfilesystem:DescribeReplicationConfigurations elasticfilesystem:ModifyMountTargetSecurityGroups elasticfilesystem:PutAccountPreferences elasticfilesystem:PutBackupPolicy

Service prefix	Actions
	elasticfilesystem:PutFileSystemPolicy elasticfilesystem:PutLifecycleConfiguration elasticfilesystem:UpdateFileSystem elasticfilesystem:UpdateFileSystemProtection

Service prefix	Actions
elasticloadbalancing	elasticloadbalancing:AddListenerCertificates
	elasticloadbalancing:AddTrustStoreRevocations
	elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
	elasticloadbalancing:AttachLoadBalancerToSubnets
	elasticloadbalancing:ConfigureHealthCheck
	elasticloadbalancing:CreateAppCookieStickinessPolicy
	elasticloadbalancing:CreateLBCookieStickinessPolicy
	elasticloadbalancing:CreateListener
	elasticloadbalancing:CreateLoadBalancer
	elasticloadbalancing:CreateLoadBalancerListeners
	elasticloadbalancing:CreateLoadBalancerPolicy
	elasticloadbalancing:CreateRule
	elasticloadbalancing:CreateTargetGroup
	elasticloadbalancing:CreateTrustStore
	elasticloadbalancing:DeleteListener
	elasticloadbalancing:DeleteLoadBalancer
	elasticloadbalancing:DeleteLoadBalancerListeners
	elasticloadbalancing:DeleteLoadBalancerPolicy
	elasticloadbalancing:DeleteRule
	elasticloadbalancing:DeleteSharedTrustStoreAssociation
	elasticloadbalancing:DeleteTargetGroup

Service prefix	Actions
	elasticloadbalancing:DeleteTrustStore
	elasticloadbalancing:DeregisterInstancesFromLoadBalancer
	elasticloadbalancing:DeregisterTargets
	elasticloadbalancing:DescribeAccountLimits
	elasticloadbalancing:DescribeCapacityReservation
	elasticloadbalancing:DescribeInstanceHealth
	elasticloadbalancing:DescribeListenerAttributes
	elasticloadbalancing:DescribeListenerCertificates
	elasticloadbalancing:DescribeListeners
	elasticloadbalancing:DescribeLoadBalancerAttributes
	elasticloadbalancing:DescribeLoadBalancerPolicies
	elasticloadbalancing:DescribeLoadBalancerPolicyTypes
	elasticloadbalancing:DescribeLoadBalancers
	elasticloadbalancing:DescribeRules
	elasticloadbalancing:DescribeSSLPolicies
	elasticloadbalancing:DescribeTargetGroupAttributes
	elasticloadbalancing:DescribeTargetGroups
	elasticloadbalancing:DescribeTargetHealth
	elasticloadbalancing:DescribeTrustStoreAssociations
	elasticloadbalancing:DescribeTrustStoreRevocations
	elasticloadbalancing:DescribeTrustStores

Service prefix	Actions
	elasticloadbalancing:DetachLoadBalancerFromSubnets
	elasticloadbalancing:DisableAvailabilityZonesForLoadBalancer
	elasticloadbalancing:EnableAvailabilityZonesForLoadBalancer
	elasticloadbalancing:GetResourcePolicy
	elasticloadbalancing:GetTrustStoreCaCertificatesBundle
	elasticloadbalancing:GetTrustStoreRevocationContent
	elasticloadbalancing:ModifyCapacityReservation
	elasticloadbalancing:ModifyIpPools
	elasticloadbalancing:ModifyListener
	elasticloadbalancing:ModifyLoadBalancerAttributes
	elasticloadbalancing:ModifyRule
	elasticloadbalancing:ModifyTargetGroup
	elasticloadbalancing:ModifyTargetGroupAttributes
	elasticloadbalancing:ModifyTrustStore
	elasticloadbalancing:RegisterInstancesWithLoadBalancer
	elasticloadbalancing:RegisterTargets
	elasticloadbalancing:RemoveListenerCertificates
	elasticloadbalancing:RemoveTrustStoreRevocations
	elasticloadbalancing:SetIpAddressType
	elasticloadbalancing:SetLoadBalancerListenerSSLCertificate
	elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer

Service prefix	Actions
	elasticloadbalancing:SetLoadBalancerPoliciesOfListener elasticloadbalancing:SetRulePriorities elasticloadbalancing:SetSecurityGroups elasticloadbalancing:SetSubnets

Service prefix	Actions
emr-containers	emr-containers:CancelJobRun emr-containers:CreateJobTemplate emr-containers:CreateManagedEndpoint emr-containers:CreateSecurityConfiguration emr-containers:CreateVirtualCluster emr-containers:DeleteJobTemplate emr-containers:DeleteManagedEndpoint emr-containers:DeleteSecurityConfiguration emr-containers:DeleteVirtualCluster emr-containers:DescribeJobRun emr-containers:DescribeJobTemplate emr-containers:DescribeManagedEndpoint emr-containers:DescribeSecurityConfiguration emr-containers:DescribeVirtualCluster emr-containers:GetManagedEndpointSessionCredentials emr-containers:ListJobRuns emr-containers:ListJobTemplates emr-containers:ListManagedEndpoints emr-containers:ListSecurityConfigurations emr-containers:ListVirtualClusters emr-containers:StartJobRun

Service prefix	Actions
emr-serverless	emr-serverless:CancelJobRun
	emr-serverless:CreateApplication
	emr-serverless:DeleteApplication
	emr-serverless:GetApplication
	emr-serverless:GetDashboardForJobRun
	emr-serverless:GetJobRun
	emr-serverless:GetResourceDashboard
	emr-serverless:GetSession
	emr-serverless:GetSessionEndpoint
	emr-serverless:ListApplications
	emr-serverless:ListJobRunAttempts
	emr-serverless:ListJobRuns
	emr-serverless:ListSessions
	emr-serverless:StartApplication
	emr-serverless:StartJobRun
	emr-serverless:StopApplication
	emr-serverless:TerminateSession
	emr-serverless:UpdateApplication

Service prefix	Actions
es	es:AcceptInboundConnection es:AcceptInboundCrossClusterSearchConnection es:AssociatePackage es:AuthorizeVpcEndpointAccess es:CancelElasticsearchServiceSoftwareUpdate es:CancelServiceSoftwareUpdate es:CreateDomain es:CreateElasticsearchDomain es:CreateIndex es:CreateOutboundConnection es:CreateOutboundCrossClusterSearchConnection es:CreatePackage es:CreateVpcEndpoint es:DeleteDomain es:DeleteElasticsearchDomain es:DeleteElasticsearchServiceRole es:DeleteInboundConnection es:DeleteInboundCrossClusterSearchConnection es:DeleteIndex es:DeleteOutboundConnection es:DeleteOutboundCrossClusterSearchConnection

Service prefix	Actions
	es:DeletePackage es:DeleteVpcEndpoint es:DescribeDomain es:DescribeDomainAutoTunes es:DescribeDomainChangeProgress es:DescribeDomainConfig es:DescribeDomainHealth es:DescribeDomainNodes es:DescribeDomains es:DescribeDryRunProgress es:DescribeElasticsearchDomain es:DescribeElasticsearchDomainConfig es:DescribeElasticsearchDomains es:DescribeElasticsearchInstanceTypeLimits es:DescribeInboundConnections es:DescribeInboundCrossClusterSearchConnections es:DescribeInstanceTypeLimits es:DescribeOutboundConnections es:DescribeOutboundCrossClusterSearchConnections es:DescribePackages es:DescribeReservedElasticsearchInstanceOfferings

Service prefix	Actions
	es:DescribeReservedElasticsearchInstances es:DescribeReservedInstanceOfferings es:DescribeReservedInstances es:DescribeVpcEndpoints es:DissociatePackage es:DissociatePackages es:GetCompatibleElasticsearchVersions es:GetCompatibleVersions es:GetDataSource es:GetDomainMaintenanceStatus es:GetPackageVersionHistory es:GetUpgradeHistory es:GetUpgradeStatus es:ListDataSources es:ListDomainNames es:ListDomainsForPackage es:ListElasticsearchInstanceTypes es:ListElasticsearchVersions es:ListInsights es:ListInstanceTypeDetails es:ListPackagesForDomain

Service prefix	Actions
	es:ListScheduledActions es:ListVersions es:ListVpcEndpointAccess es:ListVpcEndpoints es:ListVpcEndpointsForDomain es:PurchaseReservedElasticsearchInstanceOffering es:PurchaseReservedInstanceOffering es:RejectInboundConnection es:RejectInboundCrossClusterSearchConnection es:RevokeVpcEndpointAccess es:RollbackServiceSoftwareUpdate es:StartDomainMaintenance es:StartElasticsearchServiceSoftwareUpdate es:StartServiceSoftwareUpdate es:UpdateDataSource es:UpdateDomainConfig es:UpdateElasticsearchDomainConfig es:UpdateIndex es:UpdatePackage es:UpdatePackageScope es:UpdateScheduledAction

Service prefix	Actions
	es:UpdateVpcEndpoint es:UpgradeDomain es:UpgradeElasticsearchDomain

Service prefix	Actions
events	events:ActivateEventSource
	events:CancelReplay
	events:CreateApiDestination
	events:CreateArchive
	events:CreateConnection
	events:CreateEndpoint
	events:CreateEventBus
	events:CreatePartnerEventSource
	events:DeactivateEventSource
	events:DeauthorizeConnection
	events:DeleteApiDestination
	events:DeleteArchive
	events:DeleteConnection
	events:DeleteEndpoint
	events:DeleteEventBus
	events:DeletePartnerEventSource
	events:DeleteRule
	events:DescribeApiDestination
	events:DescribeArchive
	events:DescribeConnection
	events:DescribeEndpoint

Service prefix	Actions
	events:DescribeEventBus
	events:DescribeEventSource
	events:DescribePartnerEventSource
	events:DescribeReplay
	events:DescribeRule
	events:DisableRule
	events:EnableRule
	events:ListApiDestinations
	events:ListArchives
	events:ListConnections
	events:ListEndpoints
	events:ListEventBuses
	events:ListEventSources
	events:ListPartnerEventSourceAccounts
	events:ListPartnerEventSources
	events:ListReplays
	events:ListRuleNamesByTarget
	events:ListRules
	events:ListTargetsByRule
	events:PutPermission
	events:PutRule

Service prefix	Actions
	events:PutTargets
	events:RemovePermission
	events:RemoveTargets
	events:StartReplay
	events:TestEventPattern
	events:UpdateApiDestination
	events:UpdateArchive
	events:UpdateConnection
	events:UpdateEndpoint
	events:UpdateEventBus

Service prefix	Actions
finspace	finspace:CreateEnvironment finspace:CreateKxChangeset finspace:CreateKxCluster finspace:CreateKxDatabase finspace:CreateKxDataview finspace:CreateKxEnvironment finspace:CreateKxScalingGroup finspace:CreateKxUser finspace:CreateKxVolume finspace:CreateUser finspace:DeleteEnvironment finspace:DeleteKxCluster finspace:DeleteKxClusterNode finspace:DeleteKxDatabase finspace:DeleteKxDataview finspace:DeleteKxEnvironment finspace:DeleteKxScalingGroup finspace:DeleteKxUser finspace:DeleteKxVolume finspace:GetEnvironment finspace:GetKxChangeset

Service prefix	Actions
	finspace:GetKxCluster finspace:GetKxConnectionString finspace:GetKxDatabase finspace:GetKxDataview finspace:GetKxEnvironment finspace:GetKxScalingGroup finspace:GetKxUser finspace:GetKxVolume finspace:GetLoadSampleDataSetGroupIntoEnvironmentStatus finspace:GetUser finspace:ListEnvironments finspace:ListKxChangesets finspace:ListKxClusterNodes finspace:ListKxClusters finspace:ListKxDatabases finspace:ListKxDataviews finspace:ListKxEnvironments finspace:ListKxScalingGroups finspace:ListKxUsers finspace:ListKxVolumes finspace:ListUsers

Service prefix	Actions
	finspace:LoadSampleDataSetGroupIntoEnvironment finspace:ResetUserPassword finspace:UpdateEnvironment finspace:UpdateKxClusterCodeConfiguration finspace:UpdateKxClusterDatabases finspace:UpdateKxDatabase finspace:UpdateKxDataview finspace:UpdateKxEnvironment finspace:UpdateKxEnvironmentNetwork finspace:UpdateKxUser finspace:UpdateKxVolume finspace:UpdateUser
firehose	firehose:CreateDeliveryStream firehose:DeleteDeliveryStream firehose:DescribeDeliveryStream firehose:ListDeliveryStreams firehose:StartDeliveryStreamEncryption firehose:StopDeliveryStreamEncryption firehose:UpdateDestination

Service prefix	Actions
fis	fis:CreateExperimentTemplate fis:CreateTargetAccountConfiguration fis:DeleteExperimentTemplate fis:DeleteTargetAccountConfiguration fis:GetAction fis:GetExperiment fis:GetExperimentTargetAccountConfiguration fis:GetExperimentTemplate fis:GetSafetyLever fis:GetTargetAccountConfiguration fis:GetTargetResourceType fis:ListActions fis:ListExperimentResolvedTargets fis:ListExperimentTargetAccountConfigurations fis:ListExperimentTemplates fis:ListExperiments fis:ListTargetAccountConfigurations fis:ListTargetResourceTypes fis:StartExperiment fis:StopExperiment fis:UpdateExperimentTemplate

Service prefix	Actions
	fis:UpdateSafetyLeverState fis:UpdateTargetAccountConfiguration

Service prefix	Actions
fms	fms:AssociateAdminAccount
	fms:AssociateThirdPartyFirewall
	fms:BatchAssociateResource
	fms:BatchDisassociateResource
	fms:DeleteAppsList
	fms:DeleteNotificationChannel
	fms:DeletePolicy
	fms:DeleteProtocolsList
	fms:DeleteResourceSet
	fms:DisassociateAdminAccount
	fms:DisassociateThirdPartyFirewall
	fms:GetAdminAccount
	fms:GetAdminScope
	fms:GetAppsList
	fms:GetComplianceDetail
	fms:GetNotificationChannel
	fms:GetPolicy
	fms:GetProtectionStatus
	fms:GetProtocolsList
	fms:GetResourceSet
	fms:GetThirdPartyFirewallAssociationStatus

Service prefix	Actions
	fms:GetViolationDetails
	fms:ListAdminAccountsForOrganization
	fms:ListAdminsManagingAccount
	fms:ListAppsLists
	fms:ListComplianceStatus
	fms:ListDiscoveredResources
	fms:ListMemberAccounts
	fms:ListPolicies
	fms:ListProtocolsLists
	fms:ListResourceSetResources
	fms:ListResourceSets
	fms:ListThirdPartyFirewallFirewallPolicies
	fms:PutAdminAccount
	fms:PutAppsList
	fms:PutNotificationChannel
	fms:PutPolicy
	fms:PutProtocolsList
	fms:PutResourceSet

Service prefix	Actions
frauddetector	frauddetector:BatchCreateVariable frauddetector:BatchGetVariable frauddetector:CancelBatchImportJob frauddetector:CancelBatchPredictionJob frauddetector:CreateBatchImportJob frauddetector:CreateBatchPredictionJob frauddetector:CreateDetectorVersion frauddetector:CreateList frauddetector:CreateModel frauddetector:CreateModelVersion frauddetector:CreateRule frauddetector:CreateVariable frauddetector:DeleteBatchImportJob frauddetector:DeleteBatchPredictionJob frauddetector:DeleteDetector frauddetector:DeleteDetectorVersion frauddetector:DeleteEntityType frauddetector:DeleteEvent frauddetector:DeleteEventType frauddetector:DeleteEventsByEventType frauddetector:DeleteExternalModel

Service prefix	Actions
	frauddetector:DeleteLabel frauddetector:DeleteList frauddetector:DeleteModel frauddetector:DeleteModelVersion frauddetector:DeleteOutcome frauddetector:DeleteRule frauddetector:DeleteVariable frauddetector:DescribeDetector frauddetector:DescribeModelVersions frauddetector:GetBatchImportJobs frauddetector:GetBatchPredictionJobs frauddetector:GetDeleteEventsByEventTypeStatus frauddetector:GetDetectorVersion frauddetector:GetDetectors frauddetector:GetEntityTypes frauddetector:GetEvent frauddetector:GetEventPrediction frauddetector:GetEventPredictionMetadata frauddetector:GetEventTypes frauddetector:GetExternalModels frauddetector:GetKMSEncryptionKey

Service prefix	Actions
	frauddetector:GetLabels frauddetector:GetListElements frauddetector:GetListsMetadata frauddetector:GetModelVersion frauddetector:GetModels frauddetector:GetOutcomes frauddetector:GetRules frauddetector:GetVariables frauddetector:ListEventPredictions frauddetector:PutDetector frauddetector:PutEntityType frauddetector:PutEventType frauddetector:PutExternalModel frauddetector:PutKMSEncryptionKey frauddetector:PutLabel frauddetector:PutOutcome frauddetector:SendEvent frauddetector:UpdateDetectorVersion frauddetector:UpdateDetectorVersionMetadata frauddetector:UpdateDetectorVersionStatus frauddetector:UpdateEventLabel

Service prefix	Actions
	frauddetector:UpdateList frauddetector:UpdateModel frauddetector:UpdateModelVersion frauddetector:UpdateModelVersionStatus frauddetector:UpdateRuleMetadata frauddetector:UpdateRuleVersion frauddetector:UpdateVariable

Service prefix	Actions
fsx	fsx:AssociateFileSystemAliases
	fsx:CancelDataRepositoryTask
	fsx:CopyBackup
	fsx:CreateDataRepositoryTask
	fsx:CreateFileCache
	fsx:CreateFileSystem
	fsx:CreateFileSystemFromBackup
	fsx:CreateSnapshot
	fsx:CreateStorageVirtualMachine
	fsx:CreateVolume
	fsx:CreateVolumeFromBackup
	fsx:DeleteBackup
	fsx:DeleteFileCache
	fsx:DeleteFileSystem
	fsx:DeleteSnapshot
	fsx:DeleteStorageVirtualMachine
	fsx:DeleteVolume
	fsx:DescribeBackups
	fsx:DescribeDataRepositoryAssociations
	fsx:DescribeDataRepositoryTasks
	fsx:DescribeFileCaches

Service prefix	Actions
	fsx:DescribeFileSystemAliases
	fsx:DescribeFileSystems
	fsx:DescribeS3AccessPointAttachments
	fsx:DescribeSharedVpcConfiguration
	fsx:DescribeSnapshots
	fsx:DescribeStorageVirtualMachines
	fsx:DescribeVolumes
	fsx:DetachAndDeleteS3AccessPoint
	fsx:DisassociateFileSystemAliases
	fsx:ReleaseFileSystemNfsV3Locks
	fsx:RestoreVolumeFromSnapshot
	fsx:StartMisconfiguredStateRecovery
	fsx:UpdateDataRepositoryAssociation
	fsx:UpdateFileCache
	fsx:UpdateFileSystem
	fsx:UpdateSharedVpcConfiguration
	fsx:UpdateSnapshot
	fsx:UpdateStorageVirtualMachine
	fsx:UpdateVolume

Service prefix	Actions
gamelift	gamelift:AcceptMatch gamelift:ClaimGameServer gamelift:CreateAlias gamelift:CreateBuild gamelift:CreateContainerGroupDefinition gamelift:CreateFleet gamelift:CreateFleetLocations gamelift:CreateGameServerGroup gamelift:CreateGameSession gamelift:CreateGameSessionQueue gamelift:CreateLocation gamelift:CreateMatchmakingConfiguration gamelift:CreateMatchmakingRuleSet gamelift:CreatePlayerSession gamelift:CreatePlayerSessions gamelift:CreateScript gamelift:CreateVpcPeeringAuthorization gamelift:CreateVpcPeeringConnection gamelift:DeleteAlias gamelift:DeleteBuild gamelift:DeleteContainerGroupDefinition

Service prefix	Actions
	gamelift:DeleteFleet gamelift:DeleteFleetLocations gamelift:DeleteGameServerGroup gamelift:DeleteGameSessionQueue gamelift:DeleteLocation gamelift:DeleteMatchmakingConfiguration gamelift:DeleteMatchmakingRuleSet gamelift:DeleteScalingPolicy gamelift:DeleteScript gamelift:DeleteVpcPeeringAuthorization gamelift:DeleteVpcPeeringConnection gamelift:DeregisterCompute gamelift:DeregisterGameServer gamelift:DescribeAlias gamelift:DescribeBuild gamelift:DescribeCompute gamelift:DescribeContainerFleet gamelift:DescribeContainerGroupDefinition gamelift:DescribeContainerGroupPortMappings gamelift:DescribeEC2InstanceLimits gamelift:DescribeFleetAttributes

Service prefix	Actions
	gamelift:DescribeFleetCapacity gamelift:DescribeFleetEvents gamelift:DescribeFleetLocationAttributes gamelift:DescribeFleetLocationCapacity gamelift:DescribeFleetLocationUtilization gamelift:DescribeFleetPortSettings gamelift:DescribeFleetUtilization gamelift:DescribeGameServer gamelift:DescribeGameServerGroup gamelift:DescribeGameServerInstances gamelift:DescribeGameSessionDetails gamelift:DescribeGameSessionPlacement gamelift:DescribeGameSessionQueues gamelift:DescribeGameSessions gamelift:DescribeInstances gamelift:DescribeMatchmaking gamelift:DescribeMatchmakingConfigurations gamelift:DescribeMatchmakingRuleSets gamelift:DescribePlayerSessions gamelift:DescribeRuntimeConfiguration gamelift:DescribeScalingPolicies

Service prefix	Actions
	gamelift:DescribeScript gamelift:DescribeVpcPeeringAuthorizations gamelift:DescribeVpcPeeringConnections gamelift:GetComputeAccess gamelift:GetComputeAuthToken gamelift:GetGameSessionLogUrl gamelift:GetInstanceAccess gamelift:GetPlayerConnectionDetails gamelift:ListAliases gamelift:ListBuilds gamelift:ListCompute gamelift:ListContainerFleets gamelift:ListContainerGroupDefinitionVersions gamelift:ListContainerGroupDefinitions gamelift:ListFleetDeployments gamelift:ListFleets gamelift:ListGameServerGroups gamelift:ListGameServers gamelift:ListLocations gamelift:ListScripts gamelift:PutScalingPolicy

Service prefix	Actions
	gamelift:RegisterCompute gamelift:RegisterGameServer gamelift:RequestUploadCredentials gamelift:ResolveAlias gamelift:ResumeGameServerGroup gamelift:SearchGameSessions gamelift:StartFleetActions gamelift:StartGameSessionPlacement gamelift:StartMatchBackfill gamelift:StartMatchmaking gamelift:StopFleetActions gamelift:StopGameSessionPlacement gamelift:StopMatchmaking gamelift:SuspendGameServerGroup gamelift:TerminateGameSession gamelift:UpdateAlias gamelift:UpdateBuild gamelift:UpdateContainerGroupDefinition gamelift:UpdateFleetAttributes gamelift:UpdateFleetCapacity gamelift:UpdateFleetPortSettings

Service prefix	Actions
	gamelift:UpdateGameServer gamelift:UpdateGameServerGroup gamelift:UpdateGameSession gamelift:UpdateGameSessionQueue gamelift:UpdateMatchmakingConfiguration gamelift:UpdateRuntimeConfiguration gamelift:UpdateScript gamelift:ValidateMatchmakingRuleSet

Service prefix	Actions
geo	geo:AssociateTrackerConsumer
	geo:BatchDeleteDevicePositionHistory
	geo:BatchDeleteGeofence
	geo:BatchEvaluateGeofences
	geo:BatchGetDevicePosition
	geo:BatchPutGeofence
	geo:BatchUpdateDevicePosition
	geo:CalculateRoute
	geo:CalculateRouteMatrix
	geo:CancelJob
	geo:CreateGeofenceCollection
	geo:CreateMap
	geo:CreatePlaceIndex
	geo:CreateRouteCalculator
	geo:CreateTracker
	geo:DeleteGeofenceCollection
	geo:DeleteKey
	geo:DeleteMap
	geo:DeletePlaceIndex
	geo:DeleteRouteCalculator
	geo:DeleteTracker

Service prefix	Actions
	geo:DescribeGeofenceCollection
	geo:DescribeKey
	geo:DescribeMap
	geo:DescribePlaceIndex
	geo:DescribeRouteCalculator
	geo:DescribeTracker
	geo:DisassociateTrackerConsumer
	geo:ForecastGeofenceEvents
	geo:GetDevicePosition
	geo:GetDevicePositionHistory
	geo:GetGeofence
	geo:GetJob
	geo:GetMapGlyphs
	geo:GetMapSprites
	geo:GetMapStyleDescriptor
	geo:GetMapTile
	geo:GetPlace
	geo:ListDevicePositions
	geo:ListGeofenceCollections
	geo:ListGeofences
	geo:ListJobs

Service prefix	Actions
	geo:ListKeys
	geo:ListMaps
	geo:ListPlaceIndexes
	geo:ListRouteCalculators
	geo:ListTrackerConsumers
	geo:ListTrackers
	geo:PutGeofence
	geo:SearchPlaceIndexForPosition
	geo:SearchPlaceIndexForSuggestions
	geo:SearchPlaceIndexForText
	geo:UpdateGeofenceCollection
	geo:UpdateKey
	geo:UpdateMap
	geo:UpdatePlaceIndex
	geo:UpdateRouteCalculator
	geo:UpdateTracker
	geo:VerifyDevicePosition

Service prefix	Actions
glacier	glacier:AbortMultipartUpload glacier:AbortVaultLock glacier:CompleteMultipartUpload glacier:CompleteVaultLock glacier:CreateVault glacier:DeleteArchive glacier:DeleteVault glacier:DeleteVaultAccessPolicy glacier:DeleteVaultNotifications glacier:DescribeJob glacier:DescribeVault glacier:GetDataRetrievalPolicy glacier:GetJobOutput glacier:GetVaultAccessPolicy glacier:GetVaultLock glacier:GetVaultNotifications glacier:InitiateJob glacier:InitiateMultipartUpload glacier:InitiateVaultLock glacier:ListJobs glacier:ListMultipartUploads

Service prefix	Actions
	glacier:ListParts glacier:ListProvisionedCapacity glacier:ListVaults glacier:PurchaseProvisionedCapacity glacier:SetDataRetrievalPolicy glacier:SetVaultAccessPolicy glacier:SetVaultNotifications glacier:UploadArchive glacier:UploadMultipartPart

Service prefix	Actions
grafana	grafana:AssociateLicense
	grafana:CreateWorkspace
	grafana:CreateWorkspaceApiKey
	grafana:CreateWorkspaceServiceAccount
	grafana:CreateWorkspaceServiceAccountToken
	grafana:DeleteWorkspace
	grafana:DeleteWorkspaceApiKey
	grafana:DeleteWorkspaceServiceAccount
	grafana:DeleteWorkspaceServiceAccountToken
	grafana:DescribeWorkspace
	grafana:DescribeWorkspaceAuthentication
	grafana:DescribeWorkspaceConfiguration
	grafana:DisassociateLicense
	grafana:ListPermissions
	grafana:ListVersions
	grafana:ListWorkspaceServiceAccountTokens
	grafana:ListWorkspaceServiceAccounts
	grafana:ListWorkspaces
	grafana:UpdatePermissions
	grafana:UpdateWorkspace
	grafana:UpdateWorkspaceAuthentication

Service prefix	Actions
	grafana:UpdateWorkspaceConfiguration

Service prefix	Actions
greengrass	greengrass:AssociateRoleToGroup greengrass:AssociateServiceRoleToAccount greengrass:BatchAssociateClientDeviceWithCoreDevice greengrass:BatchDisassociateClientDeviceFromCoreDevice greengrass:CancelDeployment greengrass:CreateComponentVersion greengrass:CreateConnectorDefinition greengrass:CreateConnectorDefinitionVersion greengrass:CreateCoreDefinition greengrass:CreateCoreDefinitionVersion greengrass:CreateDeployment greengrass:CreateDeviceDefinition greengrass:CreateDeviceDefinitionVersion greengrass:CreateFunctionDefinition greengrass:CreateFunctionDefinitionVersion greengrass:CreateGroup greengrass:CreateGroupCertificateAuthority greengrass:CreateGroupVersion greengrass:CreateLoggerDefinition greengrass:CreateLoggerDefinitionVersion greengrass:CreateResourceDefinition

Service prefix	Actions
	greengrass:CreateResourceDefinitionVersion greengrass:CreateSoftwareUpdateJob greengrass:CreateSubscriptionDefinition greengrass:CreateSubscriptionDefinitionVersion greengrass:DeleteComponent greengrass:DeleteConnectorDefinition greengrass:DeleteCoreDefinition greengrass:DeleteCoreDevice greengrass:DeleteDeployment greengrass:DeleteDeviceDefinition greengrass:DeleteFunctionDefinition greengrass:DeleteGroup greengrass:DeleteLoggerDefinition greengrass:DeleteResourceDefinition greengrass:DeleteSubscriptionDefinition greengrass:DescribeComponent greengrass:DisassociateRoleFromGroup greengrass:DisassociateServiceRoleFromAccount greengrass:GetAssociatedRole greengrass:GetBulkDeploymentStatus greengrass:GetComponent

Service prefix	Actions
	greengrass:GetComponentVersionArtifact greengrass:GetConnectivityInfo greengrass:GetConnectorDefinition greengrass:GetConnectorDefinitionVersion greengrass:GetCoreDefinition greengrass:GetCoreDefinitionVersion greengrass:GetCoreDevice greengrass:GetDeployment greengrass:GetDeploymentStatus greengrass:GetDeviceDefinition greengrass:GetDeviceDefinitionVersion greengrass:GetFunctionDefinition greengrass:GetFunctionDefinitionVersion greengrass:GetGroup greengrass:GetGroupCertificateAuthority greengrass:GetGroupCertificateConfiguration greengrass:GetGroupVersion greengrass:GetLoggerDefinition greengrass:GetLoggerDefinitionVersion greengrass:GetResourceDefinition greengrass:GetResourceDefinitionVersion

Service prefix	Actions
	greengrass:GetServiceRoleForAccount greengrass:GetSubscriptionDefinition greengrass:GetSubscriptionDefinitionVersion greengrass:GetThingRuntimeConfiguration greengrass:ListBulkDeploymentDetailedReports greengrass:ListBulkDeployments greengrass:ListClientDevicesAssociatedWithCoreDevice greengrass:ListComponentVersions greengrass:ListComponents greengrass:ListConnectorDefinitionVersions greengrass:ListConnectorDefinitions greengrass:ListCoreDefinitionVersions greengrass:ListCoreDefinitions greengrass:ListCoreDevices greengrass:ListDeployments greengrass:ListDeviceDefinitionVersions greengrass:ListDeviceDefinitions greengrass:ListEffectiveDeployments greengrass:ListFunctionDefinitionVersions greengrass:ListFunctionDefinitions greengrass:ListGroupCertificateAuthorities

Service prefix	Actions
	greengrass:ListGroupVersions greengrass:ListGroups greengrass:ListInstalledComponents greengrass:ListLoggerDefinitionVersions greengrass:ListLoggerDefinitions greengrass:ListResourceDefinitionVersions greengrass:ListResourceDefinitions greengrass:ListSubscriptionDefinitionVersions greengrass:ListSubscriptionDefinitions greengrass:ResetDeployments greengrass:StartBulkDeployment greengrass:StopBulkDeployment greengrass:UpdateConnectivityInfo greengrass:UpdateConnectorDefinition greengrass:UpdateCoreDefinition greengrass:UpdateDeviceDefinition greengrass:UpdateFunctionDefinition greengrass:UpdateGroup greengrass:UpdateGroupCertificateConfiguration greengrass:UpdateLoggerDefinition greengrass:UpdateResourceDefinition

Service prefix	Actions
	greengrass:UpdateSubscriptionDefinition greengrass:UpdateThingRuntimeConfiguration

Service prefix	Actions
groundstation	groundstation:CancelContact
	groundstation:CreateConfig
	groundstation:CreateDataflowEndpointGroup
	groundstation:CreateDataflowEndpointGroupV2
	groundstation:CreateEphemeris
	groundstation:CreateMissionProfile
	groundstation:DeleteConfig
	groundstation:DeleteDataflowEndpointGroup
	groundstation:DeleteEphemeris
	groundstation:DeleteMissionProfile
	groundstation:DescribeContact
	groundstation:DescribeContactVersion
	groundstation:DescribeEphemeris
	groundstation:GetConfig
	groundstation:GetDataflowEndpointGroup
	groundstation:GetMinuteUsage
	groundstation:GetMissionProfile
	groundstation:GetSatellite
	groundstation:ListAntennas
	groundstation:ListConfigs
	groundstation:ListContacts

Service prefix	Actions
	groundstation:ListDataflowEndpointGroups groundstation:ListEphemerides groundstation:ListGroundStationReservations groundstation:ListGroundStations groundstation:ListMissionProfiles groundstation:ListSatellites groundstation:RegisterAgent groundstation:ReserveContact groundstation:UpdateAgentStatus groundstation:UpdateConfig groundstation:UpdateEphemeris groundstation:UpdateMissionProfile

Service prefix	Actions
guardduty	guardduty:AcceptAdministratorInvitation guardduty:AcceptInvitation guardduty:ArchiveFindings guardduty:CreateDetector guardduty:CreateFilter guardduty:CreateIPSet guardduty:CreateInvestigation guardduty:CreateMalwareProtectionPlan guardduty:CreateMembers guardduty:CreatePublishingDestination guardduty:CreateSampleFindings guardduty:CreateThreatEntitySet guardduty:CreateThreatIntelSet guardduty:CreateTrustedEntitySet guardduty:DeclineInvitations guardduty:DeleteDetector guardduty:DeleteFilter guardduty:DeleteIPSet guardduty:DeleteInvitations guardduty:DeleteMalwareProtectionPlan guardduty:DeleteMembers

Service prefix	Actions
	guardduty:DeletePublishingDestination guardduty:DeleteThreatEntitySet guardduty:DeleteThreatIntelSet guardduty:DeleteTrustedEntitySet guardduty:DescribeMalwareScans guardduty:DescribeOrganizationConfiguration guardduty:DescribePublishingDestination guardduty:DisableOrganizationAdminAccount guardduty:DisassociateFromAdministratorAccount guardduty:DisassociateFromMasterAccount guardduty:DisassociateMembers guardduty:EnableOrganizationAdminAccount guardduty:GetAdministratorAccount guardduty:GetCoverageStatistics guardduty:GetDetector guardduty:GetFilter guardduty:GetFindings guardduty:GetFindingsStatistics guardduty:GetIPSet guardduty:GetInvestigation guardduty:GetInvitationsCount

Service prefix	Actions
	guardduty:GetMalwareProtectionPlan guardduty:GetMalwareScan guardduty:GetMalwareScanSettings guardduty:GetMasterAccount guardduty:GetMemberDetectors guardduty:GetMembers guardduty:GetOrganizationStatistics guardduty:GetRemainingFreeTrialDays guardduty:GetThreatEntitySet guardduty:GetThreatIntelSet guardduty:GetTrustedEntitySet guardduty:GetUsageStatistics guardduty:InviteMembers guardduty:ListCoverage guardduty:ListDetectors guardduty:ListFilters guardduty:ListFindings guardduty:ListIPSets guardduty:ListInvestigations guardduty:ListInvitations guardduty:ListMalwareProtectionPlans

Service prefix	Actions
	guardduty:ListMalwareScans guardduty:ListMembers guardduty:ListOrganizationAdminAccounts guardduty:ListPublishingDestinations guardduty:ListThreatEntitySets guardduty:ListThreatIntelSets guardduty:ListTrustedEntitySets guardduty:StartMalwareScan guardduty:StartMonitoringMembers guardduty:StopMonitoringMembers guardduty:UnarchiveFindings guardduty:UpdateDetector guardduty:UpdateFilter guardduty:UpdateFindingsFeedback guardduty:UpdateIPSet guardduty:UpdateMalwareProtectionPlan guardduty:UpdateMalwareScanSettings guardduty:UpdateMemberDetectors guardduty:UpdateOrganizationConfiguration guardduty:UpdatePublishingDestination guardduty:UpdateThreatEntitySet

Service prefix	Actions
	guardduty:UpdateThreatIntelSet guardduty:UpdateTrustedEntitySet

Service prefix	Actions
healthlake	healthlake:CancelFHIRExportJobWithDelete
	healthlake:CreateDataTransformationProfile
	healthlake:CreateFHIRDatastore
	healthlake:CreateResource
	healthlake:DeleteDataTransformationProfile
	healthlake:DeleteFHIRDatastore
	healthlake:DeleteResource
	healthlake:DescribeDataTransformationJob
	healthlake:DescribeFHIRDatastore
	healthlake:DescribeFHIRExportJob
	healthlake:DescribeFHIRExportJobWithGet
	healthlake:DescribeFHIRImportJob
	healthlake:GetCapabilities
	healthlake:GetDataTransformationProfile
	healthlake:ListDataTransformationJobs
	healthlake:ListDataTransformationProfileVersions
	healthlake:ListDataTransformationProfiles
	healthlake:ListFHIRDatastores
	healthlake:ListFHIRExportJobs
	healthlake:ListFHIRImportJobs
	healthlake:PublishDataTransformationProfile

Service prefix	Actions
	healthlake:ReadResource
	healthlake:SearchEverything
	healthlake:SearchWithGet
	healthlake:SearchWithPost
	healthlake:StartFHIRExportJob
	healthlake:StartFHIRExportJobWithPost
	healthlake:StartFHIRImportJob
	healthlake:UpdateDataTransformationProfile
	healthlake:UpdateFHIRDatastore
	healthlake:UpdateProfileWithAgent
	healthlake:UpdateResource

Service prefix	Actions
honeycode	honeycode:BatchCreateTableRows honeycode:BatchDeleteTableRows honeycode:BatchUpdateTableRows honeycode:BatchUpsertTableRows honeycode:DescribeTableDataImportJob honeycode:GetScreenData honeycode:InvokeScreenAutomation honeycode:ListTableColumns honeycode:ListTableRows honeycode:ListTables honeycode:QueryTableRows honeycode:StartTableDataImportJob

Service prefix	Actions
iam	iam:AddClientIDToOpenIDConnectProvider
	iam:AddRoleToInstanceProfile
	iam:AddUserToGroup
	iam:AttachGroupPolicy
	iam:AttachRolePolicy
	iam:AttachUserPolicy
	iam:ChangePassword
	iam:CreateAccessKey
	iam:CreateAccountAlias
	iam:CreateGroup
	iam:CreateInstanceProfile
	iam:CreateLoginProfile
	iam:CreateOpenIDConnectProvider
	iam:CreatePolicy
	iam:CreatePolicyVersion
	iam:CreateRole
	iam:CreateSAMLProvider
	iam:CreateServiceLinkedRole
	iam:CreateServiceSpecificCredential
	iam:CreateUser
	iam:CreateVirtualMFADevice

Service prefix	Actions
	iam:DeactivateMFADevice
	iam:DeleteAccessKey
	iam:DeleteAccountAlias
	iam:DeleteAccountPasswordPolicy
	iam:DeleteCloudFrontPublicKey
	iam:DeleteGroup
	iam:DeleteGroupPolicy
	iam:DeleteInstanceProfile
	iam:DeleteLoginProfile
	iam:DeleteOpenIDConnectProvider
	iam:DeletePolicy
	iam:DeletePolicyVersion
	iam:DeleteRole
	iam:DeleteRolePermissionsBoundary
	iam:DeleteRolePolicy
	iam:DeleteSAMLProvider
	iam:DeleteSSHPublicKey
	iam:DeleteServerCertificate
	iam:DeleteServiceLinkedRole
	iam:DeleteServiceSpecificCredential
	iam:DeleteSigningCertificate

Service prefix	Actions
	iam:DeleteUser
	iam:DeleteUserPermissionsBoundary
	iam:DeleteUserPolicy
	iam:DeleteVirtualMFADevice
	iam:DetachGroupPolicy
	iam:DetachRolePolicy
	iam:DetachUserPolicy
	iam:DisableOrganizationsRootCredentialsManagement
	iam:DisableOrganizationsRootSessions
	iam:DisableOutboundWebIdentityFederation
	iam:EnableMFADevice
	iam:EnableOrganizationsRootCredentialsManagement
	iam:EnableOrganizationsRootSessions
	iam:EnableOutboundWebIdentityFederation
	iam:GenerateCredentialReport
	iam:GenerateOrganizationsAccessReport
	iam:GenerateServiceLastAccessedDetails
	iam:GetAccessKeyLastUsed
	iam:GetAccountAuthorizationDetails
	iam:GetAccountEmailAddress
	iam:GetAccountName

Service prefix	Actions
	iam:GetAccountPasswordPolicy
	iam:GetAccountSummary
	iam:GetCloudFrontPublicKey
	iam:GetContextKeysForCustomPolicy
	iam:GetContextKeysForPrincipalPolicy
	iam:GetCredentialReport
	iam:GetGroup
	iam:GetGroupPolicy
	iam:GetInstanceProfile
	iam:GetLoginProfile
	iam:GetMFADevice
	iam:GetOpenIDConnectProvider
	iam:GetOrganizationsAccessReport
	iam:GetOutboundWebIdentityFederationInfo
	iam:GetPolicy
	iam:GetPolicyVersion
	iam:GetRole
	iam:GetRolePolicy
	iam:GetSAMLProvider
	iam:GetSSHPublicKey
	iam:GetServerCertificate

Service prefix	Actions
	iam:GetServiceLastAccessedDetails
	iam:GetServiceLastAccessedDetailsWithEntities
	iam:GetServiceLinkedRoleDeletionStatus
	iam:GetUser
	iam:GetUserPolicy
	iam:ListAccessKeys
	iam:ListAccountAliases
	iam:ListAttachedGroupPolicies
	iam:ListAttachedRolePolicies
	iam:ListAttachedUserPolicies
	iam:ListCloudFrontPublicKeys
	iam:ListDelegationRequests
	iam:ListEntitiesForPolicy
	iam:ListGroupPolicies
	iam:ListGroups
	iam:ListGroupsForUser
	iam:ListInstanceProfiles
	iam:ListInstanceProfilesForRole
	iam:ListMFADevices
	iam:ListOpenIDConnectProviders
	iam:ListOrganizationsFeatures

Service prefix	Actions
	iam:ListPolicies
	iam:ListPoliciesGrantingServiceAccess
	iam:ListPolicyVersions
	iam:ListRolePolicies
	iam:ListRoles
	iam:ListSAMLProviders
	iam:ListSSHPublicKeys
	iam:ListSTSRegionalEndpointsStatus
	iam:ListServerCertificates
	iam:ListServiceSpecificCredentials
	iam:ListSigningCertificates
	iam:ListUserPolicies
	iam:ListUsers
	iam:ListVirtualMFADevices
	iam:PutGroupPolicy
	iam:PutRolePermissionsBoundary
	iam:PutRolePolicy
	iam:PutUserPermissionsBoundary
	iam:PutUserPolicy
	iam:RemoveClientIDFromOpenIDConnectProvider
	iam:RemoveRoleFromInstanceProfile

Service prefix	Actions
	iam:RemoveUserFromGroup
	iam:ResetServiceSpecificCredential
	iam:ResyncMFADevice
	iam:SetDefaultPolicyVersion
	iam:SetSTSRegionalEndpointStatus
	iam:SetSecurityTokenServicePreferences
	iam:SimulateCustomPolicy
	iam:SimulatePrincipalPolicy
	iam:UpdateAccessKey
	iam:UpdateAccountEmailAddress
	iam:UpdateAccountName
	iam:UpdateAccountPasswordPolicy
	iam:UpdateAssumeRolePolicy
	iam:UpdateCloudFrontPublicKey
	iam:UpdateGroup
	iam:UpdateLoginProfile
	iam:UpdateOpenIDConnectProviderThumbprint
	iam:UpdateRole
	iam:UpdateRoleDescription
	iam:UpdateSAMLProvider
	iam:UpdateSSHPublicKey

Service prefix	Actions
	iam:UpdateServerCertificate iam:UpdateServiceSpecificCredential iam:UpdateSigningCertificate iam:UpdateUser iam:UploadCloudFrontPublicKey iam:UploadSSHPublicKey iam:UploadServerCertificate iam:UploadSigningCertificate

Service prefix	Actions
identitystore	identitystore:CreateGroup identitystore:CreateGroupMembership identitystore:CreateUser identitystore:DeleteGroup identitystore:DeleteGroupMembership identitystore:DeleteUser identitystore:DescribeGroup identitystore:DescribeGroupMembership identitystore:DescribeUser identitystore:GetGroupId identitystore:GetGroupMembershipId identitystore:GetUserId identitystore:IsMemberInGroups identitystore:ListGroupMemberships identitystore:ListGroupMembershipsForMember identitystore:ListGroups identitystore:ListUsers identitystore:UpdateGroup identitystore:UpdateUser

Service prefix	Actions
imagebuilder	imagebuilder:CancelImageCreation
	imagebuilder:CancelLifecycleExecution
	imagebuilder:CreateComponent
	imagebuilder:CreateContainerRecipe
	imagebuilder:CreateDistributionConfiguration
	imagebuilder:CreateImage
	imagebuilder:CreateImagePipeline
	imagebuilder:CreateImageRecipe
	imagebuilder:CreateInfrastructureConfiguration
	imagebuilder:CreateLifecyclePolicy
	imagebuilder:CreateWorkflow
	imagebuilder:DeleteComponent
	imagebuilder:DeleteContainerRecipe
	imagebuilder:DeleteDistributionConfiguration
	imagebuilder:DeleteImage
	imagebuilder:DeleteImagePipeline
	imagebuilder:DeleteImageRecipe
	imagebuilder:DeleteInfrastructureConfiguration
	imagebuilder:DeleteLifecyclePolicy
	imagebuilder:DeleteWorkflow
	imagebuilder:DistributeImage

Service prefix	Actions
	imagebuilder:GetComponentPolicy
	imagebuilder:GetContainerRecipePolicy
	imagebuilder:GetImagePolicy
	imagebuilder:GetImageRecipePolicy
	imagebuilder:GetLifecycleExecution
	imagebuilder:GetLifecyclePolicy
	imagebuilder:GetMarketplaceResource
	imagebuilder:GetWorkflowExecution
	imagebuilder:GetWorkflowStepExecution
	imagebuilder:ImportComponent
	imagebuilder:ImportDiskImage
	imagebuilder:ImportVmImage
	imagebuilder:ListComponentBuildVersions
	imagebuilder:ListComponents
	imagebuilder:ListContainerRecipes
	imagebuilder:ListDistributionConfigurations
	imagebuilder:ListImageBuildVersions
	imagebuilder:ListImagePackages
	imagebuilder:ListImagePipelineImages
	imagebuilder:ListImagePipelines
	imagebuilder:ListImageRecipes

Service prefix	Actions
	imagebuilder:ListImageScanFindingAggregations
	imagebuilder:ListImageScanFindings
	imagebuilder:ListImages
	imagebuilder:ListInfrastructureConfigurations
	imagebuilder:ListLifecycleExecutionResources
	imagebuilder:ListLifecycleExecutions
	imagebuilder:ListLifecyclePolicies
	imagebuilder:ListWaitingWorkflowSteps
	imagebuilder:ListWorkflowExecutions
	imagebuilder:ListWorkflowStepExecutions
	imagebuilder:ListWorkflows
	imagebuilder:PutComponentPolicy
	imagebuilder:PutContainerRecipePolicy
	imagebuilder:PutImagePolicy
	imagebuilder:PutImageRecipePolicy
	imagebuilder:RetryImage
	imagebuilder:SendWorkflowStepAction
	imagebuilder:StartImagePipelineExecution
	imagebuilder:StartResourceStateUpdate
	imagebuilder:UpdateDistributionConfiguration
	imagebuilder:UpdateImagePipeline

Service prefix	Actions
	imagebuilder:UpdateInfrastructureConfiguration

Service prefix	Actions
inspector	inspector:AddAttributesToFindings inspector:CreateAssessmentTarget inspector:CreateAssessmentTemplate inspector:CreateExclusionsPreview inspector:CreateResourceGroup inspector:DeleteAssessmentRun inspector:DeleteAssessmentTarget inspector:DeleteAssessmentTemplate inspector:DescribeAssessmentRuns inspector:DescribeAssessmentTargets inspector:DescribeAssessmentTemplates inspector:DescribeCrossAccountAccessRole inspector:DescribeExclusions inspector:DescribeFindings inspector:DescribeResourceGroups inspector:DescribeRulesPackages inspector:GetAssessmentReport inspector:GetExclusionsPreview inspector:GetTelemetryMetadata inspector:ListAssessmentRunAgents inspector:ListAssessmentRuns

Service prefix	Actions
	inspector:ListAssessmentTargets inspector:ListAssessmentTemplates inspector:ListEventSubscriptions inspector:ListExclusions inspector:ListFindings inspector:ListRulesPackages inspector:PreviewAgents inspector:RegisterCrossAccountAccessRole inspector:RemoveAttributesFromFindings inspector:StartAssessmentRun inspector:StopAssessmentRun inspector:SubscribeToEvent inspector:UnsubscribeFromEvent inspector:UpdateAssessmentTarget

Service prefix	Actions
inspector2	inspector2:AssociateMember
	inspector2:BatchGetAccountStatus
	inspector2:BatchGetCodeSnippet
	inspector2:BatchGetFindingDetails
	inspector2:BatchGetFreeTrialInfo
	inspector2:BatchGetMemberEc2DeepInspectionStatus
	inspector2:BatchUpdateMemberEc2DeepInspectionStatus
	inspector2:CancelFindingsReport
	inspector2:CancelSbomExport
	inspector2:CreateCisScanConfiguration
	inspector2:CreateCodeSecurityIntegration
	inspector2:CreateConnector
	inspector2:CreateFilter
	inspector2:CreateFindingsReport
	inspector2:CreateSbomExport
	inspector2:DeleteCisScanConfiguration
	inspector2:DeleteCodeSecurityIntegration
	inspector2:DeleteConnector
	inspector2:DeleteFilter
	inspector2:DescribeOrganizationConfiguration
	inspector2:Disable

Service prefix	Actions
	inspector2:DisableDelegatedAdminAccount
	inspector2:DisassociateMember
	inspector2:Enable
	inspector2:EnableDelegatedAdminAccount
	inspector2:GetCisScanReport
	inspector2:GetCisScanResultDetails
	inspector2:GetClustersForImage
	inspector2:GetCodeSecurityIntegration
	inspector2:GetCodeSecurityScan
	inspector2:GetConfiguration
	inspector2:GetDelegatedAdminAccount
	inspector2:GetEc2DeepInspectionConfiguration
	inspector2:GetEncryptionKey
	inspector2:GetFindingsReportStatus
	inspector2:GetMember
	inspector2:GetSbomExport
	inspector2:ListAccountPermissions
	inspector2:ListCisScanConfigurations
	inspector2:ListCisScanResultsAggregatedByChecks
	inspector2:ListCisScanResultsAggregatedByTargetResource
	inspector2:ListCisScans

Service prefix	Actions
	inspector2:ListCodeSecurityIntegrations
	inspector2:ListCodeSecurityScanConfigurations
	inspector2:ListConnectorScanConfigurations
	inspector2:ListConnectors
	inspector2:ListCoverage
	inspector2:ListCoverageStatistics
	inspector2:ListDelegatedAdminAccounts
	inspector2:ListFilters
	inspector2:ListFindingAggregations
	inspector2:ListFindings
	inspector2:ListMembers
	inspector2:ListUsageTotals
	inspector2:ResetEncryptionKey
	inspector2:SearchVulnerabilities
	inspector2:SendCisSessionHealth
	inspector2:SendCisSessionTelemetry
	inspector2:StartCisSession
	inspector2:StartCodeSecurityScan
	inspector2:StopCisSession
	inspector2:UpdateCisScanConfiguration
	inspector2:UpdateCodeSecurityIntegration

Service prefix	Actions
	inspector2:UpdateConfiguration inspector2:UpdateConnector inspector2:UpdateEc2DeepInspectionConfiguration inspector2:UpdateEncryptionKey inspector2:UpdateFilter inspector2:UpdateOrgEc2DeepInspectionConfiguration inspector2:UpdateOrganizationConfiguration

Service prefix	Actions
iot	iot:AcceptCertificateTransfer iot:AddThingToBillingGroup iot:AddThingToThingGroup iot:AssociateSbomWithPackageVersion iot:AssociateTargetsWithJob iot:AttachPolicy iot:AttachPrincipalPolicy iot:AttachSecurityProfile iot:AttachThingPrincipal iot:CancelAuditMitigationActionsTask iot:CancelAuditTask iot:CancelCertificateTransfer iot:CancelDetectMitigationActionsTask iot:CancelJob iot:CancelJobExecution iot:ClearDefaultAuthorizer iot:ConfirmTopicRuleDestination iot:CreateAuditSuppression iot:CreateAuthorizer iot:CreateBillingGroup iot:CreateCertificateFromCsr

Service prefix	Actions
	iot:CreateCertificateProvider iot:CreateCommand iot:CreateCustomMetric iot:CreateDimension iot:CreateDomainConfiguration iot:CreateDynamicThingGroup iot:CreateFleetMetric iot:CreateJob iot:CreateJobTemplate iot:CreateKeysAndCertificate iot:CreateMitigationAction iot:CreateOTAUpdate iot:CreatePackage iot:CreatePackageVersion iot:CreatePolicy iot:CreatePolicyVersion iot:CreateProvisioningClaim iot:CreateProvisioningTemplate iot:CreateProvisioningTemplateVersion iot:CreateRoleAlias iot:CreateScheduledAudit

Service prefix	Actions
	iot:CreateSecurityProfile iot:CreateStream iot:CreateThing iot:CreateThingGroup iot:CreateThingType iot:CreateTopicRule iot:CreateTopicRuleDestination iot:DeleteAccountAuditConfiguration iot:DeleteAuditSuppression iot:DeleteAuthorizer iot:DeleteBillingGroup iot:DeleteCACertificate iot:DeleteCertificate iot:DeleteCertificateProvider iot:DeleteCommand iot:DeleteCustomMetric iot:DeleteDimension iot:DeleteDomainConfiguration iot:DeleteDynamicThingGroup iot:DeleteFleetMetric iot:DeleteJob

Service prefix	Actions
	iot:DeleteJobExecution iot:DeleteJobTemplate iot:DeleteMitigationAction iot:DeleteOTAUpdate iot:DeletePackage iot:DeletePackageVersion iot:DeletePolicy iot:DeletePolicyVersion iot:DeleteProvisioningTemplate iot:DeleteProvisioningTemplateVersion iot:DeleteRegistrationCode iot:DeleteRoleAlias iot:DeleteScheduledAudit iot:DeleteSecurityProfile iot:DeleteStream iot:DeleteThing iot:DeleteThingGroup iot:DeleteThingType iot:DeleteTopicRule iot:DeleteTopicRuleDestination iot:DeleteV2LoggingLevel

Service prefix	Actions
	iot:DeprecateThingType iot:DescribeAccountAuditConfiguration iot:DescribeAuditFinding iot:DescribeAuditMitigationActionsTask iot:DescribeAuditSuppression iot:DescribeAuditTask iot:DescribeAuthorizer iot:DescribeBillingGroup iot:DescribeCACertificate iot:DescribeCertificate iot:DescribeCertificateProvider iot:DescribeCustomMetric iot:DescribeDefaultAuthorizer iot:DescribeDetectMitigationActionsTask iot:DescribeDimension iot:DescribeDomainConfiguration iot:DescribeEncryptionConfiguration iot:DescribeEndpoint iot:DescribeEventConfigurations iot:DescribeFleetMetric iot:DescribeIndex

Service prefix	Actions
	iot:DescribeJob iot:DescribeJobExecution iot:DescribeJobTemplate iot:DescribeManagedJobTemplate iot:DescribeMitigationAction iot:DescribeProvisioningTemplate iot:DescribeProvisioningTemplateVersion iot:DescribeRoleAlias iot:DescribeScheduledAudit iot:DescribeSecurityProfile iot:DescribeStream iot:DescribeThing iot:DescribeThingGroup iot:DescribeThingRegistrationTask iot:DescribeThingType iot:DetachPolicy iot:DetachPrincipalPolicy iot:DetachSecurityProfile iot:DetachThingPrincipal iot:DisableTopicRule iot:DisassociateSbomFromPackageVersion

Service prefix	Actions
	iot:EnableTopicRule iot:GetBehaviorModelTrainingSummaries iot:GetBucketsAggregation iot:GetCardinality iot:GetCommand iot:GetEffectivePolicies iot:GetJobDocument iot:GetLoggingOptions iot:GetOTAUpdate iot:GetPackage iot:GetPackageConfiguration iot:GetPackageVersion iot:GetPercentiles iot:GetPolicy iot:GetPolicyVersion iot:GetRegistrationCode iot:GetStatistics iot:GetThingConnectivityData iot:GetTopicRule iot:GetTopicRuleDestination iot:GetV2LoggingOptions

Service prefix	Actions
	iot:ListActiveViolations iot:ListAttachedPolicies iot:ListAuditFindings iot:ListAuditMitigationActionsExecutions iot:ListAuditMitigationActionsTasks iot:ListAuditSuppressions iot:ListAuditTasks iot:ListAuthorizers iot:ListBillingGroups iot:ListCACertificates iot:ListCertificateProviders iot:ListCertificates iot:ListCertificatesByCA iot:ListCommands iot:ListCustomMetrics iot:ListDetectMitigationActionsExecutions iot:ListDetectMitigationActionsTasks iot:ListDimensions iot:ListDomainConfigurations iot:ListFleetMetrics iot:ListIndices

Service prefix	Actions
	iot:ListJobExecutionsForJob iot:ListJobExecutionsForThing iot:ListJobTemplates iot:ListJobs iot:ListManagedJobTemplates iot:ListMetricValues iot:ListMitigationActions iot:ListOTAUpdates iot:ListOutgoingCertificates iot:ListPackageVersions iot:ListPackages iot:ListPolicies iot:ListPolicyPrincipals iot:ListPolicyVersions iot:ListPrincipalPolicies iot:ListPrincipalThings iot:ListPrincipalThingsV2 iot:ListProvisioningTemplateVersions iot:ListProvisioningTemplates iot:ListRelatedResourcesForAuditFinding iot:ListRoleAliases

Service prefix	Actions
	iot:ListSbomValidationResults iot:ListScheduledAudits iot:ListSecurityProfiles iot:ListSecurityProfilesForTarget iot:ListStreams iot:ListTargetsForPolicy iot:ListTargetsForSecurityProfile iot:ListThingGroups iot:ListThingGroupsForThing iot:ListThingPrincipals iot:ListThingPrincipalsV2 iot:ListThingRegistrationTaskReports iot:ListThingRegistrationTasks iot:ListThingTypes iot:ListThings iot:ListThingsInBillingGroup iot:ListThingsInThingGroup iot:ListTopicRuleDestinations iot:ListTopicRules iot:ListV2LoggingLevels iot:ListViolationEvents

Service prefix	Actions
	iot:PutVerificationStateOnViolation iot:RegisterCACertificate iot:RegisterCertificate iot:RegisterCertificateWithoutCA iot:RegisterThing iot:RejectCertificateTransfer iot:RemoveThingFromBillingGroup iot:RemoveThingFromThingGroup iot:ReplaceTopicRule iot:SearchIndex iot:SetDefaultAuthorizer iot:SetDefaultPolicyVersion iot:SetLoggingOptions iot:SetV2LoggingLevel iot:SetV2LoggingOptions iot:StartAuditMitigationActionsTask iot:StartDetectMitigationActionsTask iot:StartOnDemandAuditTask iot:StartThingRegistrationTask iot:StopThingRegistrationTask iot:TestAuthorization

Service prefix	Actions
	iot:TestInvokeAuthorizer iot:TransferCertificate iot:UpdateAccountAuditConfiguration iot:UpdateAuditSuppression iot:UpdateAuthorizer iot:UpdateBillingGroup iot:UpdateCACertificate iot:UpdateCertificate iot:UpdateCertificateProvider iot:UpdateCommand iot:UpdateCustomMetric iot:UpdateDimension iot:UpdateDomainConfiguration iot:UpdateDynamicThingGroup iot:UpdateEncryptionConfiguration iot:UpdateEventConfigurations iot:UpdateFleetMetric iot:UpdateIndexingConfiguration iot:UpdateJob iot:UpdateMitigationAction iot:UpdatePackage

Service prefix	Actions
	iot:UpdatePackageConfiguration iot:UpdatePackageVersion iot:UpdateProvisioningTemplate iot:UpdateRoleAlias iot:UpdateScheduledAudit iot:UpdateSecurityProfile iot:UpdateStream iot:UpdateThing iot:UpdateThingGroup iot:UpdateThingGroupsForThing iot:UpdateThingType iot:UpdateTopicRuleDestination iot:ValidateSecurityProfileBehaviors

Service prefix	Actions
iotanalytics	iotanalytics:CancelPipelineReprocessing
	iotanalytics:CreateChannel
	iotanalytics:CreateDataset
	iotanalytics:CreateDatasetContent
	iotanalytics:CreateDatastore
	iotanalytics:CreatePipeline
	iotanalytics:DeleteChannel
	iotanalytics:DeleteDataset
	iotanalytics:DeleteDatasetContent
	iotanalytics:DeleteDatastore
	iotanalytics:DeletePipeline
	iotanalytics:DescribeChannel
	iotanalytics:DescribeDataset
	iotanalytics:DescribeDatastore
	iotanalytics:DescribeLoggingOptions
	iotanalytics:DescribePipeline
	iotanalytics:GetDatasetContent
	iotanalytics:ListChannels
	iotanalytics:ListDatasetContents
	iotanalytics:ListDatasets
	iotanalytics:ListDatastores

Service prefix	Actions
	iotanalytics:ListPipelines iotanalytics:PutLoggingOptions iotanalytics:RunPipelineActivity iotanalytics:SampleChannelData iotanalytics:StartPipelineReprocessing iotanalytics:UpdateChannel iotanalytics:UpdateDataset iotanalytics:UpdateDatastore iotanalytics:UpdatePipeline
iotdeviceadvisor	iotdeviceadvisor:CreateSuiteDefinition iotdeviceadvisor:DeleteSuiteDefinition iotdeviceadvisor:GetEndpoint iotdeviceadvisor:GetSuiteDefinition iotdeviceadvisor:GetSuiteRun iotdeviceadvisor:GetSuiteRunReport iotdeviceadvisor:ListSuiteDefinitions iotdeviceadvisor:ListSuiteRuns iotdeviceadvisor:StartSuiteRun iotdeviceadvisor:StopSuiteRun iotdeviceadvisor:UpdateSuiteDefinition

Service prefix	Actions
iotevents	iotevents:BatchAcknowledgeAlarm
	iotevents:BatchDeleteDetector
	iotevents:BatchDisableAlarm
	iotevents:BatchEnableAlarm
	iotevents:BatchResetAlarm
	iotevents:BatchSnoozeAlarm
	iotevents:BatchUpdateDetector
	iotevents:CreateAlarmModel
	iotevents:CreateDetectorModel
	iotevents:CreateInput
	iotevents:DeleteAlarmModel
	iotevents:DeleteDetectorModel
	iotevents:DeleteInput
	iotevents:DescribeAlarm
	iotevents:DescribeAlarmModel
	iotevents:DescribeDetector
	iotevents:DescribeDetectorModel
	iotevents:DescribeDetectorModelAnalysis
	iotevents:DescribeInput
	iotevents:DescribeLoggingOptions
	iotevents:GetDetectorModelAnalysisResults

Service prefix	Actions
	iotevents:ListAlarmModelVersions iotevents:ListAlarmModels iotevents:ListAlarms iotevents:ListDetectorModelVersions iotevents:ListDetectorModels iotevents:ListDetectors iotevents:ListInputRoutings iotevents:ListInputs iotevents:PutLoggingOptions iotevents:StartDetectorModelAnalysis iotevents:UpdateAlarmModel iotevents:UpdateDetectorModel iotevents:UpdateInput
iotfleethub	iotfleethub:CreateApplication iotfleethub:DeleteApplication iotfleethub:DescribeApplication iotfleethub:ListApplications iotfleethub:UpdateApplication

Service prefix	Actions
iotsitewise	iotsitewise:AssociateAssets
	iotsitewise:AssociateTimeSeriesToAssetProperty
	iotsitewise:BatchAssociateProjectAssets
	iotsitewise:BatchDisassociateProjectAssets
	iotsitewise:CreateAccessPolicy
	iotsitewise:CreateAsset
	iotsitewise:CreateAssetModel
	iotsitewise:CreateAssetModelCompositeModel
	iotsitewise:CreateBulkImportJob
	iotsitewise:CreateComputationModel
	iotsitewise:CreateDashboard
	iotsitewise:CreateDataset
	iotsitewise:CreateGateway
	iotsitewise:CreatePortal
	iotsitewise:CreateProject
	iotsitewise:DeleteAccessPolicy
	iotsitewise:DeleteAsset
	iotsitewise:DeleteAssetModel
	iotsitewise:DeleteAssetModelCompositeModel
	iotsitewise:DeleteComputationModel
	iotsitewise:DeleteDashboard

Service prefix	Actions
	iotsitewise:DeleteDataset
	iotsitewise:DeleteGateway
	iotsitewise:DeletePortal
	iotsitewise:DeleteProject
	iotsitewise:DeleteTimeSeries
	iotsitewise:DescribeAccessPolicy
	iotsitewise:DescribeAsset
	iotsitewise:DescribeAssetCompositeModel
	iotsitewise:DescribeAssetModel
	iotsitewise:DescribeAssetModelCompositeModel
	iotsitewise:DescribeAssetModelInterfaceRelationship
	iotsitewise:DescribeAssetProperty
	iotsitewise:DescribeBulkImportJob
	iotsitewise:DescribeComputationModel
	iotsitewise:DescribeComputationModelExecutionSummary
	iotsitewise:DescribeDashboard
	iotsitewise:DescribeDataset
	iotsitewise:DescribeDefaultEncryptionConfiguration
	iotsitewise:DescribeExecution
	iotsitewise:DescribeGateway
	iotsitewise:DescribeGatewayCapabilityConfiguration

Service prefix	Actions
	iotsitewise:DescribeLoggingOptions
	iotsitewise:DescribePortal
	iotsitewise:DescribeProject
	iotsitewise:DescribeStorageConfiguration
	iotsitewise:DescribeTimeSeries
	iotsitewise:DisassociateAssets
	iotsitewise:DisassociateTimeSeriesFromAssetProperty
	iotsitewise:ExecuteAction
	iotsitewise:ExecuteQuery
	iotsitewise:ListAccessPolicies
	iotsitewise:ListActions
	iotsitewise:ListAssetModelCompositeModels
	iotsitewise:ListAssetModelProperties
	iotsitewise:ListAssetModels
	iotsitewise:ListAssetProperties
	iotsitewise:ListAssetRelationships
	iotsitewise:ListAssets
	iotsitewise:ListAssociatedAssets
	iotsitewise:ListBulkImportJobs
	iotsitewise:ListCompositionRelationships
	iotsitewise:ListComputationModelDataBindingUsages

Service prefix	Actions
	iotsitewise:ListComputationModelResolveToResources
	iotsitewise:ListComputationModels
	iotsitewise:ListDashboards
	iotsitewise:ListDatasets
	iotsitewise:ListExecutions
	iotsitewise:ListGateways
	iotsitewise:ListInterfaceRelationships
	iotsitewise:ListPortals
	iotsitewise:ListProjectAssets
	iotsitewise:ListProjects
	iotsitewise:ListTimeSeries
	iotsitewise:PutDefaultEncryptionConfiguration
	iotsitewise:PutLoggingOptions
	iotsitewise:PutStorageConfiguration
	iotsitewise:UpdateAccessPolicy
	iotsitewise:UpdateAsset
	iotsitewise:UpdateAssetModel
	iotsitewise:UpdateAssetModelCompositeModel
	iotsitewise:UpdateAssetProperty
	iotsitewise:UpdateComputationModel
	iotsitewise:UpdateDashboard

Service prefix	Actions
	iotsitewise:UpdateDataset iotsitewise:UpdateGateway iotsitewise:UpdateGatewayCapabilityConfiguration iotsitewise:UpdatePortal iotsitewise:UpdateProject

Service prefix	Actions
iottwinmaker	iottwinmaker:CancelMetadataTransferJob iottwinmaker:CreateComponentType iottwinmaker:CreateEntity iottwinmaker:CreateMetadataTransferJob iottwinmaker:CreateScene iottwinmaker:CreateSyncJob iottwinmaker:CreateWorkspace iottwinmaker:DeleteComponentType iottwinmaker:DeleteEntity iottwinmaker:DeleteScene iottwinmaker:DeleteSyncJob iottwinmaker:DeleteWorkspace iottwinmaker:ExecuteQuery iottwinmaker:GetMetadataTransferJob iottwinmaker:GetPricingPlan iottwinmaker:GetScene iottwinmaker:GetSyncJob iottwinmaker:ListComponentTypes iottwinmaker:ListComponents iottwinmaker:ListEntities iottwinmaker:ListMetadataTransferJobs

Service prefix	Actions
	iottwinmaker:ListProperties iottwinmaker:ListScenes iottwinmaker:ListSyncJobs iottwinmaker:ListSyncResources iottwinmaker:ListWorkspaces iottwinmaker:UpdateComponentType iottwinmaker:UpdateEntity iottwinmaker:UpdatePricingPlan iottwinmaker:UpdateScene iottwinmaker:UpdateWorkspace

Service prefix	Actions
iotwireless	iotwireless:AssociateAwsAccountWithPartnerAccount
	iotwireless:AssociateMulticastGroupWithFuotaTask
	iotwireless:AssociateWirelessDeviceWithFuotaTask
	iotwireless:AssociateWirelessDeviceWithMulticastGroup
	iotwireless:AssociateWirelessDeviceWithThing
	iotwireless:AssociateWirelessGatewayWithCertificate
	iotwireless:AssociateWirelessGatewayWithThing
	iotwireless:CancelMulticastGroupSession
	iotwireless:CreateDestination
	iotwireless:CreateDeviceProfile
	iotwireless:CreateFuotaTask
	iotwireless:CreateMulticastGroup
	iotwireless:CreateNetworkAnalyzerConfiguration
	iotwireless:CreateServiceProfile
	iotwireless:CreateWirelessDevice
	iotwireless:CreateWirelessGateway
	iotwireless:CreateWirelessGatewayTask
	iotwireless:CreateWirelessGatewayTaskDefinition
	iotwireless:DeleteDestination
	iotwireless:DeleteDeviceProfile
	iotwireless:DeleteFuotaTask

Service prefix	Actions
	iotwireless:DeleteMulticastGroup
	iotwireless:DeleteNetworkAnalyzerConfiguration
	iotwireless:DeleteQueuedMessages
	iotwireless:DeleteServiceProfile
	iotwireless:DeleteWirelessDevice
	iotwireless:DeleteWirelessDeviceImportTask
	iotwireless:DeleteWirelessGateway
	iotwireless:DeleteWirelessGatewayTask
	iotwireless:DeleteWirelessGatewayTaskDefinition
	iotwireless:DeregisterWirelessDevice
	iotwireless:DisassociateAwsAccountFromPartnerAccount
	iotwireless:DisassociateMulticastGroupFromFuotaTask
	iotwireless:DisassociateWirelessDeviceFromFuotaTask
	iotwireless:DisassociateWirelessDeviceFromMulticastGroup
	iotwireless:DisassociateWirelessDeviceFromThing
	iotwireless:DisassociateWirelessGatewayFromCertificate
	iotwireless:DisassociateWirelessGatewayFromThing
	iotwireless:GetDestination
	iotwireless:GetDeviceProfile
	iotwireless:GetEventConfigurationByResourceTypes
	iotwireless:GetFuotaTask

Service prefix	Actions
	iotwireless:GetLogLevelsByResourceTypes
	iotwireless:GetMetricConfiguration
	iotwireless:GetMetrics
	iotwireless:GetMulticastGroup
	iotwireless:GetMulticastGroupSession
	iotwireless:GetNetworkAnalyzerConfiguration
	iotwireless:GetPartnerAccount
	iotwireless:GetPosition
	iotwireless:GetPositionConfiguration
	iotwireless:GetPositionEstimate
	iotwireless:GetResourceEventConfiguration
	iotwireless:GetResourceLogLevel
	iotwireless:GetResourcePosition
	iotwireless:GetServiceEndpoint
	iotwireless:GetServiceProfile
	iotwireless:GetWirelessDevice
	iotwireless:GetWirelessDeviceImportTask
	iotwireless:GetWirelessDeviceStatistics
	iotwireless:GetWirelessGateway
	iotwireless:GetWirelessGatewayCertificate
	iotwireless:GetWirelessGatewayFirmwareInformation

Service prefix	Actions
	iotwireless:GetWirelessGatewayStatistics
	iotwireless:GetWirelessGatewayTask
	iotwireless:GetWirelessGatewayTaskDefinition
	iotwireless:ListDestinations
	iotwireless:ListDeviceProfiles
	iotwireless:ListDevicesForWirelessDeviceImportTask
	iotwireless:ListEventConfigurations
	iotwireless:ListFuotaTasks
	iotwireless:ListMulticastGroups
	iotwireless:ListMulticastGroupsByFuotaTask
	iotwireless:ListNetworkAnalyzerConfigurations
	iotwireless:ListPartnerAccounts
	iotwireless:ListPositionConfigurations
	iotwireless:ListQueuedMessages
	iotwireless:ListServiceProfiles
	iotwireless:ListWirelessDeviceImportTasks
	iotwireless:ListWirelessDevices
	iotwireless:ListWirelessGatewayTaskDefinitions
	iotwireless:ListWirelessGateways
	iotwireless:PutPositionConfiguration
	iotwireless:PutResourceLogLevel

Service prefix	Actions
	iotwireless:ResetAllResourceLogLevels
	iotwireless:ResetResourceLogLevel
	iotwireless:SendDataToMulticastGroup
	iotwireless:SendDataToWirelessDevice
	iotwireless:StartBulkAssociateWirelessDeviceWithMulticastGroup
	iotwireless:StartBulkDisassociateWirelessDeviceFromMulticastGroup
	iotwireless:StartFuotaTask
	iotwireless:StartMulticastGroupSession
	iotwireless:StartNetworkAnalyzerStream
	iotwireless:StartSingleWirelessDeviceImportTask
	iotwireless:StartWirelessDeviceImportTask
	iotwireless:TestWirelessDevice
	iotwireless:UpdateDestination
	iotwireless:UpdateEventConfigurationByResourceTypes
	iotwireless:UpdateFuotaTask
	iotwireless:UpdateLogLevelsByResourceTypes
	iotwireless:UpdateMetricConfiguration
	iotwireless:UpdateMulticastGroup
	iotwireless:UpdateNetworkAnalyzerConfiguration
	iotwireless:UpdatePartnerAccount

Service prefix	Actions
	iotwireless:UpdatePosition iotwireless:UpdateResourceEventConfiguration iotwireless:UpdateResourcePosition iotwireless:UpdateWirelessDevice iotwireless:UpdateWirelessDeviceImportTask iotwireless:UpdateWirelessGateway

Service prefix	Actions
ivs	ivs:BatchGetChannel ivs:BatchGetStreamKey ivs:BatchStartViewerSessionRevocation ivs:CreateAdConfiguration ivs:CreateChannel ivs:CreateEncoderConfiguration ivs:CreateIngestConfiguration ivs:CreateParticipantToken ivs:CreatePlaybackRestrictionPolicy ivs:CreateRecordingConfiguration ivs:CreateStorageConfiguration ivs:CreateStreamKey ivs:DeleteAdConfiguration ivs:DeleteChannel ivs:DeleteEncoderConfiguration ivs:DeleteIngestConfiguration ivs:DeletePlaybackKeyPair ivs:DeletePlaybackRestrictionPolicy ivs:DeletePublicKey ivs:DeleteRecordingConfiguration ivs:DeleteStorageConfiguration

Service prefix	Actions
	ivs:DeleteStreamKey ivs:DisconnectParticipant ivs:GetAdConfiguration ivs:GetChannel ivs:GetComposition ivs:GetEncoderConfiguration ivs:GetIngestConfiguration ivs:GetParticipant ivs:GetPlaybackKeyPair ivs:GetPlaybackRestrictionPolicy ivs:GetPublicKey ivs:GetRecordingConfiguration ivs:GetStorageConfiguration ivs:GetStream ivs:GetStreamKey ivs:GetStreamSession ivs:ImportPlaybackKeyPair ivs:ImportPublicKey ivs:InsertAdBreak ivs:ListAdConfigurations ivs:ListChannels

Service prefix	Actions
	ivs:ListCompositions ivs:ListEncoderConfigurations ivs:ListIngestConfigurations ivs:ListParticipantEvents ivs:ListParticipantReplicas ivs:ListParticipants ivs:ListPlaybackKeyPairs ivs:ListPlaybackRestrictionPolicies ivs:ListPublicKeys ivs:ListRecordingConfigurations ivs:ListStorageConfigurations ivs:ListStreamKeys ivs:ListStreamSessions ivs:ListStreams ivs:PutMetadata ivs:StartComposition ivs:StartViewerSessionRevocation ivs:StopComposition ivs:StopStream ivs:UpdateAdConfiguration ivs:UpdateChannel

Service prefix	Actions
	ivs:UpdateIngestConfiguration ivs:UpdatePlaybackRestrictionPolicy
ivschat	ivschat:CreateChatToken ivschat:CreateLoggingConfiguration ivschat:CreateRoom ivschat:DeleteLoggingConfiguration ivschat:DeleteMessage ivschat:DeleteRoom ivschat:DisconnectUser ivschat:GetLoggingConfiguration ivschat:GetRoom ivschat:ListLoggingConfigurations ivschat:ListRooms ivschat:SendEvent ivschat:UpdateLoggingConfiguration ivschat:UpdateRoom

Service prefix	Actions
kafka	kafka:BatchAssociateScramSecret kafka:BatchDisassociateScramSecret kafka:CreateCluster kafka:CreateClusterV2 kafka:CreateConfiguration kafka:CreateReplicator kafka:CreateVpcConnection kafka:DeleteCluster kafka:DeleteClusterPolicy kafka:DeleteConfiguration kafka:DeleteReplicator kafka:DeleteVpcConnection kafka:DescribeCluster kafka:DescribeClusterOperation kafka:DescribeClusterOperationV2 kafka:DescribeClusterV2 kafka:DescribeConfiguration kafka:DescribeConfigurationRevision kafka:DescribeVpcConnection kafka:GetBootstrapBrokers kafka:GetClusterPolicy

Service prefix	Actions
	kafka:GetCompatibleKafkaVersions kafka:ListClientVpcConnections kafka:ListClusterOperations kafka:ListClusterOperationsV2 kafka:ListClusters kafka:ListClustersV2 kafka:ListConfigurationRevisions kafka:ListConfigurations kafka:ListKafkaVersions kafka:ListNodes kafka:ListReplicators kafka:ListScramSecrets kafka:ListVpcConnections kafka:PutClusterPolicy kafka:RebootBroker kafka:RejectClientVpcConnection kafka:UpdateBrokerCount kafka:UpdateBrokerStorage kafka:UpdateBrokerType kafka:UpdateClusterConfiguration kafka:UpdateClusterKafkaVersion

Service prefix	Actions
	kafka:UpdateConfiguration kafka:UpdateConnectivity kafka:UpdateMonitoring kafka:UpdateRebalancing kafka:UpdateReplicationInfo kafka:UpdateSecurity kafka:UpdateStorage
kafkaconnect	kafkaconnect:CreateConnector kafkaconnect:CreateCustomPlugin kafkaconnect:CreateWorkerConfiguration kafkaconnect:DeleteConnector kafkaconnect:DeleteCustomPlugin kafkaconnect:DeleteWorkerConfiguration kafkaconnect:DescribeConnector kafkaconnect:DescribeCustomPlugin kafkaconnect:DescribeWorkerConfiguration kafkaconnect:ListConnectorOperations kafkaconnect:ListConnectors kafkaconnect:ListCustomPlugins kafkaconnect:ListWorkerConfigurations kafkaconnect:UpdateConnector

Service prefix	Actions
kendra	kendra:AssociateEntitiesToExperience kendra:AssociatePersonasToEntities kendra:BatchDeleteDocument kendra:BatchDeleteFeaturedResultsSet kendra:BatchGetDocumentStatus kendra:BatchPutDocument kendra:ClearQuerySuggestions kendra:CreateAccessControlConfiguration kendra:CreateDataSource kendra:CreateExperience kendra:CreateFaq kendra:CreateFeaturedResultsSet kendra:CreateIndex kendra:CreateQuerySuggestionsBlockList kendra:CreateThesaurus kendra:DeleteDataSource kendra:DeleteExperience kendra:DeleteFaq kendra:DeleteIndex kendra:DeletePrincipalMapping kendra:DeleteQuerySuggestionsBlockList

Service prefix	Actions
	kendra:DeleteThesaurus kendra:DescribeAccessControlConfiguration kendra:DescribeDataSource kendra:DescribeExperience kendra:DescribeFaq kendra:DescribeFeaturedResultsSet kendra:DescribeIndex kendra:DescribePrincipalMapping kendra:DescribeQuerySuggestionsBlockList kendra:DescribeQuerySuggestionsConfig kendra:DescribeThesaurus kendra:DisassociateEntitiesFromExperience kendra:DisassociatePersonasFromEntities kendra:GetQuerySuggestions kendra:GetSnapshots kendra:ListAccessControlConfigurations kendra:ListDataSourceSyncJobs kendra:ListDataSources kendra:ListEntityPersonas kendra:ListExperienceEntities kendra:ListExperiences

Service prefix	Actions
	kendra:ListFaqs kendra:ListFeaturedResultsSets kendra:ListGroupsOlderThanOrderingId kendra:ListIndices kendra:ListQuerySuggestionsBlockLists kendra:ListThesauri kendra:PutPrincipalMapping kendra:Query kendra:Retrieve kendra:StartDataSourceSyncJob kendra:StopDataSourceSyncJob kendra:SubmitFeedback kendra:UpdateDataSource kendra:UpdateExperience kendra:UpdateFeaturedResultsSet kendra:UpdateIndex kendra:UpdateQuerySuggestionsBlockList kendra:UpdateQuerySuggestionsConfig kendra:UpdateThesaurus

Service prefix	Actions
kinesis	kinesis:CreateStream
	kinesis:DecreaseStreamRetentionPeriod
	kinesis:DeleteStream
	kinesis:DeregisterStreamConsumer
	kinesis:DescribeAccountSettings
	kinesis:DescribeLimits
	kinesis:DescribeStream
	kinesis:DescribeStreamConsumer
	kinesis:DescribeStreamSummary
	kinesis:DisableEnhancedMonitoring
	kinesis:EnableEnhancedMonitoring
	kinesis:IncreaseStreamRetentionPeriod
	kinesis:ListShards
	kinesis:ListStreamConsumers
	kinesis:ListStreams
	kinesis:MergeShards
	kinesis:RegisterStreamConsumer
	kinesis:SplitShard
	kinesis:StartStreamEncryption
	kinesis:StopStreamEncryption
	kinesis:UpdateAccountSettings

Service prefix	Actions
	kinesis:UpdateShardCount kinesis:UpdateStreamMode

Service prefix	Actions
kinesisanalytics	kinesisanalytics:AddApplicationCloudWatchLoggingOption kinesisanalytics:AddApplicationInput kinesisanalytics:AddApplicationInputProcessingConfiguration kinesisanalytics:AddApplicationOutput kinesisanalytics:AddApplicationReferenceDataSource kinesisanalytics:AddApplicationVpcConfiguration kinesisanalytics:CreateApplication kinesisanalytics:CreateApplicationPresignedUrl kinesisanalytics:CreateApplicationSnapshot kinesisanalytics:DeleteApplication kinesisanalytics:DeleteApplicationCloudWatchLoggingOption kinesisanalytics:DeleteApplicationInputProcessingConfiguration kinesisanalytics:DeleteApplicationOutput kinesisanalytics:DeleteApplicationReferenceDataSource kinesisanalytics:DeleteApplicationSnapshot kinesisanalytics:DeleteApplicationVpcConfiguration kinesisanalytics:DescribeApplication kinesisanalytics:DescribeApplicationOperation kinesisanalytics:DescribeApplicationSnapshot kinesisanalytics:DescribeApplicationVersion kinesisanalytics:DiscoverInputSchema

Service prefix	Actions
	kinesisanalytics:ListApplicationOperations kinesisanalytics:ListApplicationSnapshots kinesisanalytics:ListApplicationVersions kinesisanalytics:ListApplications kinesisanalytics:RollbackApplication kinesisanalytics:StartApplication kinesisanalytics:StopApplication kinesisanalytics:UpdateApplication kinesisanalytics:UpdateApplicationMaintenanceConfiguration

Service prefix	Actions
kms	kms:CancelKeyDeletion
	kms:ConnectCustomKeyStore
	kms:CreateAlias
	kms:CreateCustomKeyStore
	kms:CreateGrant
	kms:CreateKey
	kms:Decrypt
	kms:DeleteAlias
	kms:DeleteCustomKeyStore
	kms:DeleteImportedKeyMaterial
	kms:DeriveSharedSecret
	kms:DescribeCustomKeyStores
	kms:DescribeKey
	kms:DisableKey
	kms:DisableKeyRotation
	kms:DisconnectCustomKeyStore
	kms:EnableKey
	kms:EnableKeyRotation
	kms:Encrypt
	kms:GenerateDataKey
	kms:GenerateDataKeyPair

Service prefix	Actions
	kms:GenerateDataKeyPairWithoutPlaintext
	kms:GenerateDataKeyWithoutPlaintext
	kms:GenerateMac
	kms:GenerateRandom
	kms:GetKeyLastUsage
	kms:GetKeyPolicy
	kms:GetKeyRotationStatus
	kms:GetParametersForImport
	kms:GetPublicKey
	kms:ImportKeyMaterial
	kms:ListAliases
	kms:ListGrants
	kms:ListKeyPolicies
	kms:ListKeyRotations
	kms:ListKeys
	kms:ListRetirableGrants
	kms:ReplicateKey
	kms:RetireGrant
	kms:RevokeGrant
	kms:RotateKeyOnDemand
	kms:ScheduleKeyDeletion

Service prefix	Actions
	kms:Sign kms:UpdateAlias kms:UpdateCustomKeyStore kms:UpdateKeyDescription kms:UpdatePrimaryRegion kms:Verify kms:VerifyMac

Service prefix	Actions
lambda	lambda:AddLayerVersionPermission lambda:AddPermission lambda:CreateAlias lambda:CreateCodeSigningConfig lambda:CreateEventSourceMapping lambda:CreateFunction lambda:CreateFunctionUrlConfig lambda:DeleteAlias lambda:DeleteCapacityProvider lambda:DeleteCodeSigningConfig lambda:DeleteEventSourceMapping lambda:DeleteFunction lambda:DeleteFunctionCodeSigningConfig lambda:DeleteFunctionConcurrency lambda:DeleteFunctionEventInvokeConfig lambda:DeleteFunctionUrlConfig lambda:DeleteLayerVersion lambda:DeleteProvisionedConcurrencyConfig lambda:GetAccountSettings lambda:GetAlias lambda:GetCapacityProvider

Service prefix	Actions
	lambda:GetCodeSigningConfig lambda:GetEventSourceMapping lambda:GetFunction lambda:GetFunctionCodeSigningConfig lambda:GetFunctionConcurrency lambda:GetFunctionConfiguration lambda:GetFunctionEventInvokeConfig lambda:GetFunctionRecursionConfig lambda:GetFunctionScalingConfig lambda:GetFunctionUrlConfig lambda:GetLayerVersion lambda:GetLayerVersionPolicy lambda:GetPolicy lambda:GetProvisionedConcurrencyConfig lambda:GetRuntimeManagementConfig lambda:ListAliases lambda:ListCapacityProviders lambda:ListCodeSigningConfigs lambda:ListDurableExecutionsByFunction lambda:ListEventSourceMappings lambda:ListFunctionEventInvokeConfigs

Service prefix	Actions
	lambda:ListFunctionUrlConfigs lambda:ListFunctions lambda:ListFunctionsByCodeSigningConfig lambda:ListLayerVersions lambda:ListLayers lambda:ListProvisionedConcurrencyConfigs lambda:ListVersionsByFunction lambda:PublishLayerVersion lambda:PublishVersion lambda:PutFunctionCodeSigningConfig lambda:PutFunctionConcurrency lambda:PutFunctionEventInvokeConfig lambda:PutFunctionRecursionConfig lambda:PutFunctionScalingConfig lambda:PutProvisionedConcurrencyConfig lambda:PutRuntimeManagementConfig lambda:RemoveLayerVersionPermission lambda:RemovePermission lambda:UpdateAlias lambda:UpdateCapacityProvider lambda:UpdateCodeSigningConfig

Service prefix	Actions
	lambda:UpdateEventSourceMapping lambda:UpdateFunctionCode lambda:UpdateFunctionConfiguration lambda:UpdateFunctionEventInvokeConfig lambda:UpdateFunctionUrlConfig

Service prefix	Actions
lex	lex:BatchCreateCustomVocabularyItem
	lex:BatchDeleteCustomVocabularyItem
	lex:BatchUpdateCustomVocabularyItem
	lex:BuildBotLocale
	lex:CreateBotAlias
	lex:CreateBotReplica
	lex:CreateBotVersion
	lex:CreateExport
	lex:CreateIntentVersion
	lex:CreateResourcePolicy
	lex:CreateSlotTypeVersion
	lex:CreateTestSetDiscrepancyReport
	lex:CreateUploadUrl
	lex:DeleteBot
	lex:DeleteBotAnalyzerRecommendation
	lex:DeleteBotChannelAssociation
	lex:DeleteBotReplica
	lex:DeleteExport
	lex:DeleteImport
	lex:DeleteIntentVersion
	lex:DeleteResourcePolicy

Service prefix	Actions
	lex:DeleteSlotTypeVersion
	lex:DeleteTestSet
	lex:DeleteUtterances
	lex:DescribeBotAlias
	lex:DescribeBotAnalyzerRecommendation
	lex:DescribeBotRecommendation
	lex:DescribeBotReplica
	lex:DescribeBotResourceGeneration
	lex:DescribeBotVersion
	lex:DescribeCustomVocabularyMetadata
	lex:DescribeExport
	lex:DescribeImport
	lex:DescribeResourcePolicy
	lex:DescribeTestExecution
	lex:DescribeTestSet
	lex:DescribeTestSetDiscrepancyReport
	lex:DescribeTestSetGeneration
	lex:GenerateBotElement
	lex:GetBot
	lex:GetBotAlias
	lex:GetBotAliases

Service prefix	Actions
	lex:GetBotChannelAssociation
	lex:GetBotChannelAssociations
	lex:GetBotVersions
	lex:GetBots
	lex:GetBuiltinIntent
	lex:GetBuiltinIntents
	lex:GetBuiltinSlotTypes
	lex:GetExport
	lex:GetImport
	lex:GetIntent
	lex:GetIntentVersions
	lex:GetIntents
	lex:GetMigration
	lex:GetMigrations
	lex:GetSlotType
	lex:GetSlotTypeVersions
	lex:GetSlotTypes
	lex:GetTestExecutionArtifactsUrl
	lex:GetUtterancesView
	lex:ListBotAliasReplicas
	lex:ListBotAliases

Service prefix	Actions
	lex:ListBotRecommendations
	lex:ListBotReplicas
	lex:ListBotResourceGenerations
	lex:ListBotVersionReplicas
	lex:ListBotVersions
	lex:ListBots
	lex:ListBuiltInIntents
	lex:ListBuiltInSlotTypes
	lex:ListCustomVocabularyItems
	lex:ListExports
	lex:ListImports
	lex:ListIntentMetrics
	lex:ListIntentPaths
	lex:ListRecommendedIntents
	lex:ListSessionAnalyticsData
	lex:ListSessionMetrics
	lex:ListTestExecutionResultItems
	lex:ListTestExecutions
	lex:ListTestSets
	lex:PutBot
	lex:PutBotAlias

Service prefix	Actions
	lex:PutIntent
	lex:PutSlotType
	lex:SearchAssociatedTranscripts
	lex:StartBotAnalyzer
	lex:StartBotRecommendation
	lex:StartImport
	lex:StartMigration
	lex:StartTestExecution
	lex:StartTestSetGeneration
	lex:StopBotAnalyzer
	lex:StopBotRecommendation
	lex:UpdateBotAlias
	lex:UpdateBotRecommendation
	lex:UpdateExport
	lex:UpdateResourcePolicy

Service prefix	Actions
license-manager-linux-subscriptions	license-manager-linux-subscriptions:DeregisterSubscriptionProvider license-manager-linux-subscriptions:GetRegisteredSubscriptionProvider license-manager-linux-subscriptions:GetServiceSettings license-manager-linux-subscriptions:ListLinuxSubscriptionInstances license-manager-linux-subscriptions:ListLinuxSubscriptions license-manager-linux-subscriptions:ListRegisteredSubscriptionProviders license-manager-linux-subscriptions:RegisterSubscriptionProvider license-manager-linux-subscriptions:UpdateServiceSettings

Service prefix	Actions
lightsail	lightsail:AllocateStaticIp
	lightsail:AttachCertificateToDistribution
	lightsail:AttachDisk
	lightsail:AttachInstancesToLoadBalancer
	lightsail:AttachLoadBalancerTlsCertificate
	lightsail:AttachStaticIp
	lightsail:CloseInstancePublicPorts
	lightsail:CopySnapshot
	lightsail:CreateBucket
	lightsail:CreateBucketAccessKey
	lightsail:CreateCertificate
	lightsail:CreateCloudFormationStack
	lightsail:CreateContactMethod
	lightsail:CreateContainerService
	lightsail:CreateContainerServiceDeployment
	lightsail:CreateContainerServiceRegistryLogin
	lightsail:CreateDisk
	lightsail:CreateDiskFromSnapshot
	lightsail:CreateDiskSnapshot
	lightsail:CreateDistribution
	lightsail:CreateDomain

Service prefix	Actions
	lightsail:CreateGUISessionAccessDetails
	lightsail:CreateInstanceSnapshot
	lightsail:CreateInstances
	lightsail:CreateInstancesFromSnapshot
	lightsail:CreateKeyPair
	lightsail:CreateLoadBalancer
	lightsail:CreateLoadBalancerTlsCertificate
	lightsail:CreateRelationalDatabase
	lightsail:CreateRelationalDatabaseFromSnapshot
	lightsail:CreateRelationalDatabaseSnapshot
	lightsail:DeleteAlarm
	lightsail:DeleteAutoSnapshot
	lightsail:DeleteBucket
	lightsail:DeleteBucketAccessKey
	lightsail:DeleteCertificate
	lightsail:DeleteContactMethod
	lightsail:DeleteContainerImage
	lightsail:DeleteContainerService
	lightsail:DeleteDisk
	lightsail:DeleteDiskSnapshot
	lightsail:DeleteDistribution

Service prefix	Actions
	lightsail:DeleteDomain
	lightsail:DeleteDomainEntry
	lightsail:DeleteInstance
	lightsail:DeleteInstanceSnapshot
	lightsail:DeleteKeyPair
	lightsail:DeleteKnownHostKeys
	lightsail:DeleteLoadBalancer
	lightsail:DeleteLoadBalancerTlsCertificate
	lightsail:DeleteRelationalDatabase
	lightsail:DeleteRelationalDatabaseSnapshot
	lightsail:DetachCertificateFromDistribution
	lightsail:DetachDisk
	lightsail:DetachInstancesFromLoadBalancer
	lightsail:DetachStaticIp
	lightsail:DisableAddOn
	lightsail:DownloadDefaultKeyPair
	lightsail:EnableAddOn
	lightsail:ExportSnapshot
	lightsail:GetActiveNames
	lightsail:GetAlarms
	lightsail:GetAutoSnapshots

Service prefix	Actions
	lightsail:GetBlueprints
	lightsail:GetBucketAccessKeys
	lightsail:GetBucketBundles
	lightsail:GetBucketMetricData
	lightsail:GetBuckets
	lightsail:GetBundles
	lightsail:GetCertificates
	lightsail:GetCloudFormationStackRecords
	lightsail:GetContactMethods
	lightsail:GetContainerAPIMetadata
	lightsail:GetContainerImages
	lightsail:GetContainerLog
	lightsail:GetContainerServiceDeployments
	lightsail:GetContainerServiceMetricData
	lightsail:GetContainerServicePowers
	lightsail:GetContainerServices
	lightsail:GetCostEstimate
	lightsail:GetDisk
	lightsail:GetDiskSnapshot
	lightsail:GetDiskSnapshots
	lightsail:GetDisks

Service prefix	Actions
	lightsail:GetDistributionBundles
	lightsail:GetDistributionLatestCacheReset
	lightsail:GetDistributionMetricData
	lightsail:GetDistributions
	lightsail:GetDomain
	lightsail:GetExportSnapshotRecords
	lightsail:GetInstance
	lightsail:GetInstanceMetricData
	lightsail:GetInstancePortStates
	lightsail:GetInstanceSnapshot
	lightsail:GetInstanceSnapshots
	lightsail:GetInstanceState
	lightsail:GetInstances
	lightsail:GetKeyPair
	lightsail:GetKeyPairs
	lightsail:GetLoadBalancer
	lightsail:GetLoadBalancerMetricData
	lightsail:GetLoadBalancerTlsCertificates
	lightsail:GetLoadBalancerTlsPolicies
	lightsail:GetLoadBalancers
	lightsail:GetOperation

Service prefix	Actions
	lightsail:GetOperations
	lightsail:GetOperationsForResource
	lightsail:GetRegions
	lightsail:GetRelationalDatabase
	lightsail:GetRelationalDatabaseBlueprints
	lightsail:GetRelationalDatabaseBundles
	lightsail:GetRelationalDatabaseEvents
	lightsail:GetRelationalDatabaseLogEvents
	lightsail:GetRelationalDatabaseLogStreams
	lightsail:GetRelationalDatabaseMasterUserPassword
	lightsail:GetRelationalDatabaseMetricData
	lightsail:GetRelationalDatabaseParameters
	lightsail:GetRelationalDatabaseSnapshot
	lightsail:GetRelationalDatabaseSnapshots
	lightsail:GetRelationalDatabases
	lightsail:GetSetupHistory
	lightsail:GetStaticIp
	lightsail:GetStaticIps
	lightsail:ImportKeyPair
	lightsail:IsVpcPeered
	lightsail:OpenInstancePublicPorts

Service prefix	Actions
	lightsail:PeerVpc
	lightsail:PutAlarm
	lightsail:PutInstancePublicPorts
	lightsail:RebootInstance
	lightsail:RebootRelationalDatabase
	lightsail:RegisterContainerImage
	lightsail:ReleaseStaticIp
	lightsail:ResetDistributionCache
	lightsail:SendContactMethodVerification
	lightsail:SetIpAddressType
	lightsail:SetResourceAccessForBucket
	lightsail:SetupInstanceHttps
	lightsail:StartGUISession
	lightsail:StartInstance
	lightsail:StartRelationalDatabase
	lightsail:StopGUISession
	lightsail:StopInstance
	lightsail:StopRelationalDatabase
	lightsail:TestAlarm
	lightsail:UnpeerVpc
	lightsail:UpdateBucket

Service prefix	Actions
	lightsail:UpdateBucketBundle lightsail:UpdateContainerService lightsail:UpdateDistribution lightsail:UpdateDistributionBundle lightsail:UpdateDomainEntry lightsail:UpdateInstanceMetadataOptions lightsail:UpdateLoadBalancerAttribute lightsail:UpdateRelationalDatabase lightsail:UpdateRelationalDatabaseParameters

Service prefix	Actions
logs	logs:AssociateKmsKey logs:AssociateSourceToS3TableIntegration logs:CancelExportTask logs:CancelImportTask logs:CreateDelivery logs:CreateExportTask logs:CreateLogAnomalyDetector logs:CreateLogGroup logs:CreateLogStream logs:CreateLookupTable logs:DeleteDataProtectionPolicy logs:DeleteDelivery logs:DeleteDeliveryDestination logs:DeleteDeliveryDestinationPolicy logs:DeleteDeliverySource logs:DeleteDestination logs:DeleteIndexPolicy logs:DeleteIntegration logs:DeleteLogAnomalyDetector logs:DeleteLogGroup logs:DeleteLogStream

Service prefix	Actions
	logs:DeleteLookupTable logs:DeleteMetricFilter logs:DeleteQueryDefinition logs:DeleteResourcePolicy logs:DeleteRetentionPolicy logs:DeleteScheduledQuery logs:DeleteSubscriptionFilter logs:DeleteSyslogConfiguration logs:DeleteTransformer logs:DescribeAccountPolicies logs:DescribeConfigurationTemplates logs:DescribeDeliveries logs:DescribeDeliveryDestinations logs:DescribeDeliverySources logs:DescribeDestinations logs:DescribeExportTasks logs:DescribeFieldIndexes logs:DescribeImportTaskBatches logs:DescribeImportTasks logs:DescribeIndexPolicies logs:DescribeLogGroups

Service prefix	Actions
	logs:DescribeLogStreams logs:DescribeLookupTables logs:DescribeMetricFilters logs:DescribeQueries logs:DescribeQueryDefinitions logs:DescribeResourcePolicies logs:DescribeSubscriptionFilters logs:DisassociateKmsKey logs:DisassociateSourceFromS3TableIntegration logs:GetDataProtectionPolicy logs:GetDelivery logs:GetDeliveryDestination logs:GetDeliveryDestinationPolicy logs:GetDeliverySource logs:GetIntegration logs:GetLogAnomalyDetector logs:GetLogFields logs:GetLogGroupFields logs:GetLogRecord logs:GetLookupTable logs:GetQueryResults

Service prefix	Actions
	logs:GetScheduledQuery logs:GetScheduledQueryHistory logs:GetStorageTierPolicy logs:GetTransformer logs:ListAnomalies logs:ListIntegrations logs:ListLogAnomalyDetectors logs:ListLogGroupsForQuery logs:ListScheduledQueries logs:ListSourcesForS3TableIntegration logs:ListSyslogConfigurations logs:PutBearerTokenAuthentication logs:PutDataProtectionPolicy logs:PutDeliveryDestination logs:PutDeliveryDestinationPolicy logs:PutDeliverySource logs:PutDestination logs:PutDestinationPolicy logs:PutIndexPolicy logs:PutIntegration logs:PutLogGroupDeletionProtection

Service prefix	Actions
	logs:PutMetricFilter logs:PutQueryDefinition logs:PutResourcePolicy logs:PutRetentionPolicy logs:PutStorageTierPolicy logs:PutSubscriptionFilter logs:PutSyslogConfiguration logs:PutTransformer logs:StartLiveTail logs:StartQuery logs:StopQuery logs:TestMetricFilter logs:TestTransformer logs:UpdateAnomaly logs:UpdateDeliveryConfiguration logs:UpdateLogAnomalyDetector logs:UpdateLookupTable

Service prefix	Actions
lookoutequipment	lookoutequipment:CreateDataset lookoutequipment:CreateInferenceScheduler lookoutequipment:CreateLabel lookoutequipment:CreateLabelGroup lookoutequipment:CreateModel lookoutequipment:DeleteDataset lookoutequipment:DeleteInferenceScheduler lookoutequipment:DeleteLabel lookoutequipment:DeleteLabelGroup lookoutequipment:DeleteModel lookoutequipment:DeleteResourcePolicy lookoutequipment:DeleteRetrainingScheduler lookoutequipment:DescribeDataIngestionJob lookoutequipment:DescribeDataset lookoutequipment:DescribeInferenceScheduler lookoutequipment:DescribeLabelGroup lookoutequipment:DescribeModel lookoutequipment:DescribeModelVersion lookoutequipment:DescribeResourcePolicy lookoutequipment:DescribeRetrainingScheduler lookoutequipment:Describelabel

Service prefix	Actions
	lookoutequipment:ImportDataset
	lookoutequipment:ImportModelVersion
	lookoutequipment:ListDataIngestionJobs
	lookoutequipment:ListDatasets
	lookoutequipment:ListInferenceEvents
	lookoutequipment:ListInferenceExecutions
	lookoutequipment:ListInferenceSchedulers
	lookoutequipment:ListLabelGroups
	lookoutequipment:ListLabels
	lookoutequipment:ListModelVersions
	lookoutequipment:ListModels
	lookoutequipment:ListRetrainingSchedulers
	lookoutequipment:ListSensorStatistics
	lookoutequipment:PutResourcePolicy
	lookoutequipment:StartDataIngestionJob
	lookoutequipment:StartInferenceScheduler
	lookoutequipment:StartRetrainingScheduler
	lookoutequipment:StopInferenceScheduler
	lookoutequipment:StopRetrainingScheduler
	lookoutequipment:UpdateActiveModelVersion
	lookoutequipment:UpdateInferenceScheduler

Service prefix	Actions
	lookoutequipment:UpdateLabelGroup lookoutequipment:UpdateModel lookoutequipment:UpdateRetrainingScheduler

Service prefix	Actions
lookoutmetrics	lookoutmetrics:ActivateAnomalyDetector lookoutmetrics:BackTestAnomalyDetector lookoutmetrics:CreateAlert lookoutmetrics:CreateAnomalyDetector lookoutmetrics:CreateMetricSet lookoutmetrics:DeactivateAnomalyDetector lookoutmetrics:DeleteAlert lookoutmetrics:DeleteAnomalyDetector lookoutmetrics:DescribeAlert lookoutmetrics:DescribeAnomalyDetectionExecutions lookoutmetrics:DescribeAnomalyDetector lookoutmetrics:DescribeMetricSet lookoutmetrics:DetectMetricSetConfig lookoutmetrics:GetAnomalyGroup lookoutmetrics:GetDataQualityMetrics lookoutmetrics:GetFeedback lookoutmetrics:GetSampleData lookoutmetrics:ListAlerts lookoutmetrics:ListAnomalyDetectors lookoutmetrics:ListAnomalyGroupRelatedMetrics lookoutmetrics:ListAnomalyGroupSummaries

Service prefix	Actions
	lookoutmetrics:ListAnomalyGroupTimeSeries lookoutmetrics:ListMetricSets lookoutmetrics:PutFeedback lookoutmetrics:UpdateAlert lookoutmetrics:UpdateAnomalyDetector lookoutmetrics:UpdateMetricSet

Service prefix	Actions
m2	m2:CancelBatchJobExecution
	m2:CreateApplication
	m2:CreateDataSetExportTask
	m2:CreateDataSetImportTask
	m2:CreateDeployment
	m2:CreateEnvironment
	m2:DeleteApplication
	m2:DeleteApplicationFromEnvironment
	m2:DeleteEnvironment
	m2:GetApplication
	m2:GetApplicationVersion
	m2:GetBatchJobExecution
	m2:GetDataSetDetails
	m2:GetDataSetExportTask
	m2:GetDataSetImportTask
	m2:GetDeployment
	m2:GetEnvironment
	m2:GetSignedBluinsightsUrl
	m2:ListApplicationVersions
	m2:ListApplications
	m2:ListBatchJobDefinitions

Service prefix	Actions
	m2:ListBatchJobExecutions
	m2:ListBatchJobRestartPoints
	m2:ListDataSetExportHistory
	m2:ListDataSetImportHistory
	m2:ListDataSets
	m2:ListDeployments
	m2:ListEngineVersions
	m2:ListEnvironments
	m2:StartApplication
	m2:StartBatchJob
	m2:StopApplication
	m2:UpdateApplication
	m2:UpdateEnvironment

Service prefix	Actions
managedblockchain	managedblockchain:CreateAccessor managedblockchain:CreateMember managedblockchain:CreateNetwork managedblockchain:CreateNode managedblockchain:CreateProposal managedblockchain:DeleteAccessor managedblockchain:DeleteMember managedblockchain:DeleteNode managedblockchain:GetAccessor managedblockchain:GetMember managedblockchain:GetNetwork managedblockchain:GetNode managedblockchain:GetProposal managedblockchain:InvokeRpcPolygonMainnet managedblockchain:InvokeRpcPolygonMumbaiTestnet managedblockchain:ListAccessors managedblockchain:ListInvitations managedblockchain:ListMembers managedblockchain:ListNetworks managedblockchain:ListNodes managedblockchain:ListProposalVotes

Service prefix	Actions
	managedblockchain:ListProposals managedblockchain:RejectInvitation managedblockchain:UpdateMember managedblockchain:UpdateNode managedblockchain:VoteOnProposal

Service prefix	Actions
mediaconnect	mediaconnect:AddBridgeOutputs
	mediaconnect:AddBridgeSources
	mediaconnect:AddFlowMediaStreams
	mediaconnect:AddFlowOutputs
	mediaconnect:AddFlowSources
	mediaconnect:AddFlowVpcInterfaces
	mediaconnect:CreateBridge
	mediaconnect:CreateFlow
	mediaconnect:CreateGateway
	mediaconnect:DeleteBridge
	mediaconnect:DeleteFlow
	mediaconnect:DeleteGateway
	mediaconnect:DeleteRouterInput
	mediaconnect:DeleteRouterNetworkInterface
	mediaconnect:DeleteRouterOutput
	mediaconnect:DeregisterGatewayInstance
	mediaconnect:DescribeBridge
	mediaconnect:DescribeFlow
	mediaconnect:DescribeFlowSourceMetadata
	mediaconnect:DescribeFlowSourceThumbnail
	mediaconnect:DescribeGateway

Service prefix	Actions
	mediaconnect:DescribeGatewayInstance
	mediaconnect:DescribeOffering
	mediaconnect:DescribeReservation
	mediaconnect:GetRouterInput
	mediaconnect:GetRouterInputSourceMetadata
	mediaconnect:GetRouterInputThumbnail
	mediaconnect:GetRouterNetworkInterface
	mediaconnect:GetRouterOutput
	mediaconnect:GrantFlowEntitlements
	mediaconnect:ListBridges
	mediaconnect:ListEntitlements
	mediaconnect:ListFlows
	mediaconnect:ListGatewayInstances
	mediaconnect:ListGateways
	mediaconnect:ListOfferings
	mediaconnect:ListReservations
	mediaconnect:ListRouterInputs
	mediaconnect:ListRouterNetworkInterfaces
	mediaconnect:ListRouterOutputs
	mediaconnect:PurchaseOffering
	mediaconnect:RemoveBridgeOutput

Service prefix	Actions
	mediaconnect:RemoveBridgeSource
	mediaconnect:RemoveFlowMediaStream
	mediaconnect:RemoveFlowOutput
	mediaconnect:RemoveFlowSource
	mediaconnect:RemoveFlowVpcInterface
	mediaconnect:RestartRouterInput
	mediaconnect:RestartRouterOutput
	mediaconnect:RevokeFlowEntitlement
	mediaconnect:StartFlow
	mediaconnect:StartRouterInput
	mediaconnect:StartRouterOutput
	mediaconnect:StopFlow
	mediaconnect:StopRouterInput
	mediaconnect:StopRouterOutput
	mediaconnect:TakeRouterInput
	mediaconnect:UpdateBridge
	mediaconnect:UpdateBridgeOutput
	mediaconnect:UpdateBridgeSource
	mediaconnect:UpdateBridgeState
	mediaconnect:UpdateFlow
	mediaconnect:UpdateFlowEntitlement

Service prefix	Actions
	mediaconnect:UpdateFlowMediaStream mediaconnect:UpdateGatewayInstance

Service prefix	Actions
mediaconvert	mediaconvert:AssociateCertificate mediaconvert:CancelJob mediaconvert:CreateJob mediaconvert:CreateJobTemplate mediaconvert:CreatePreset mediaconvert:CreateQueue mediaconvert:CreateResourceShare mediaconvert:DeleteJobTemplate mediaconvert:DeletePolicy mediaconvert:DeletePreset mediaconvert:DeleteQueue mediaconvert:DescribeEndpoints mediaconvert:DisassociateCertificate mediaconvert:GetJob mediaconvert:GetJobTemplate mediaconvert:GetPolicy mediaconvert:GetPreset mediaconvert:GetQueue mediaconvert:ListJobTemplates mediaconvert:ListJobs mediaconvert:ListPresets

Service prefix	Actions
	mediaconvert:ListQueues mediaconvert:ListVersions mediaconvert:Probe mediaconvert:PutPolicy mediaconvert:SearchJobs mediaconvert:UpdateJobTemplate mediaconvert:UpdatePreset mediaconvert:UpdateQueue

Service prefix	Actions
medialive	medialive:AcceptInputDeviceTransfer
	medialive:BatchDelete
	medialive:BatchStart
	medialive:BatchStop
	medialive:BatchUpdateSchedule
	medialive:CancelInputDeviceTransfer
	medialive:ClaimDevice
	medialive:CreateChannel
	medialive:CreateChannelPlacementGroup
	medialive:CreateCloudWatchAlarmTemplate
	medialive:CreateCloudWatchAlarmTemplateGroup
	medialive:CreateCluster
	medialive:CreateEventBridgeRuleTemplate
	medialive:CreateEventBridgeRuleTemplateGroup
	medialive:CreateInput
	medialive:CreateInputSecurityGroup
	medialive:CreateMultiplex
	medialive:CreateMultiplexProgram
	medialive:CreateNetwork
	medialive:CreateNode
	medialive:CreateNodeRegistrationScript

Service prefix	Actions
	medialive:CreatePartnerInput
	medialive:CreateSdiSource
	medialive:CreateSignalMap
	medialive:DeleteChannel
	medialive:DeleteChannelPlacementGroup
	medialive:DeleteCloudWatchAlarmTemplate
	medialive:DeleteCloudWatchAlarmTemplateGroup
	medialive:DeleteCluster
	medialive:DeleteEventBridgeRuleTemplate
	medialive:DeleteEventBridgeRuleTemplateGroup
	medialive:DeleteInput
	medialive:DeleteInputSecurityGroup
	medialive:DeleteMultiplex
	medialive:DeleteMultiplexProgram
	medialive:DeleteNetwork
	medialive:DeleteNode
	medialive:DeleteReservation
	medialive:DeleteSchedule
	medialive:DeleteSdiSource
	medialive:DeleteSignalMap
	medialive:DescribeAccountConfiguration

Service prefix	Actions
	medialive:DescribeChannel
	medialive:DescribeChannelPlacementGroup
	medialive:DescribeCluster
	medialive:DescribeInput
	medialive:DescribeInputDevice
	medialive:DescribeInputDeviceThumbnail
	medialive:DescribeInputSecurityGroup
	medialive:DescribeMultiplex
	medialive:DescribeMultiplexProgram
	medialive:DescribeNetwork
	medialive:DescribeNode
	medialive:DescribeOffering
	medialive:DescribeReservation
	medialive:DescribeSchedule
	medialive:DescribeSdiSource
	medialive:DescribeThumbnails
	medialive:GetCloudWatchAlarmTemplate
	medialive:GetCloudWatchAlarmTemplateGroup
	medialive:GetEventBridgeRuleTemplate
	medialive:GetEventBridgeRuleTemplateGroup
	medialive:GetSignalMap

Service prefix	Actions
	medialive:ListAlerts
	medialive:ListChannelPlacementGroups
	medialive:ListChannels
	medialive:ListCloudWatchAlarmTemplateGroups
	medialive:ListCloudWatchAlarmTemplates
	medialive:ListClusterAlerts
	medialive:ListClusters
	medialive:ListEventBridgeRuleTemplateGroups
	medialive:ListEventBridgeRuleTemplates
	medialive:ListInputDeviceTransfers
	medialive:ListInputDevices
	medialive:ListInputSecurityGroups
	medialive:ListInputs
	medialive:ListMultiplexAlerts
	medialive:ListMultiplexPrograms
	medialive:ListMultiplexes
	medialive:ListNetworks
	medialive:ListNodes
	medialive:ListOfferings
	medialive:ListReservations
	medialive:ListSdiSources

Service prefix	Actions
	medialive:ListSignalMaps
	medialive:ListVersions
	medialive:PurchaseOffering
	medialive:RebootInputDevice
	medialive:RejectInputDeviceTransfer
	medialive:RestartChannelPipelines
	medialive:StartChannel
	medialive:StartDeleteMonitorDeployment
	medialive:StartInputDevice
	medialive:StartInputDeviceMaintenanceWindow
	medialive:StartMonitorDeployment
	medialive:StartMultiplex
	medialive:StartUpdateSignalMap
	medialive:StopChannel
	medialive:StopInputDevice
	medialive:StopMultiplex
	medialive:TransferInputDevice
	medialive:UpdateAccountConfiguration
	medialive:UpdateChannel
	medialive:UpdateChannelClass
	medialive:UpdateChannelPlacementGroup

Service prefix	Actions
	medialive:UpdateCloudWatchAlarmTemplate
	medialive:UpdateCloudWatchAlarmTemplateGroup
	medialive:UpdateCluster
	medialive:UpdateEventBridgeRuleTemplate
	medialive:UpdateEventBridgeRuleTemplateGroup
	medialive:UpdateInput
	medialive:UpdateInputDevice
	medialive:UpdateInputSecurityGroup
	medialive:UpdateMultiplex
	medialive:UpdateMultiplexProgram
	medialive:UpdateNetwork
	medialive:UpdateNode
	medialive:UpdateNodeState
	medialive:UpdateReservation
	medialive:UpdateSdiSource

Service prefix	Actions
mediatailor	mediatailor:ConfigureLogsForPlaybackConfiguration mediatailor:CreateChannel mediatailor:CreateLiveSource mediatailor:CreatePrefetchSchedule mediatailor:CreateProgram mediatailor:CreateSourceLocation mediatailor:CreateVodSource mediatailor:DeleteChannel mediatailor:DeleteChannelPolicy mediatailor:DeleteLiveSource mediatailor:DeletePlaybackConfiguration mediatailor:DeletePrefetchSchedule mediatailor:DeleteProgram mediatailor:DeleteSourceLocation mediatailor:DeleteVodSource mediatailor:DescribeChannel mediatailor:DescribeLiveSource mediatailor:DescribeProgram mediatailor:DescribeSourceLocation mediatailor:DescribeVodSource mediatailor:GetChannelPolicy

Service prefix	Actions
	mediatailor:GetChannelSchedule mediatailor:GetPlaybackConfiguration mediatailor:GetPrefetchSchedule mediatailor:ListAlerts mediatailor:ListChannels mediatailor:ListLiveSources mediatailor:ListPlaybackConfigurations mediatailor:ListPrefetchSchedules mediatailor:ListSourceLocations mediatailor:ListVodSources mediatailor:PutChannelPolicy mediatailor:PutPlaybackConfiguration mediatailor:StartChannel mediatailor:StopChannel mediatailor:UpdateChannel mediatailor:UpdateLiveSource mediatailor:UpdateProgram mediatailor:UpdateSourceLocation mediatailor:UpdateVodSource

Service prefix	Actions
memorydb	memorydb:BatchUpdateCluster
	memorydb:CopySnapshot
	memorydb:CreateAcl
	memorydb:CreateCluster
	memorydb:CreateMultiRegionCluster
	memorydb:CreateParameterGroup
	memorydb:CreateSnapshot
	memorydb:CreateSubnetGroup
	memorydb:CreateUser
	memorydb:DeleteAcl
	memorydb:DeleteCluster
	memorydb:DeleteMultiRegionCluster
	memorydb:DeleteParameterGroup
	memorydb:DeleteSnapshot
	memorydb:DeleteSubnetGroup
	memorydb:DeleteUser
	memorydb:DescribeAcls
	memorydb:DescribeClusters
	memorydb:DescribeEngineVersions
	memorydb:DescribeEvents
	memorydb:DescribeMultiRegionClusters

Service prefix	Actions
	memorydb:DescribeMultiRegionParameterGroups
	memorydb:DescribeMultiRegionParameters
	memorydb:DescribeParameterGroups
	memorydb:DescribeParameters
	memorydb:DescribeReservedNodes
	memorydb:DescribeReservedNodesOfferings
	memorydb:DescribeServiceUpdates
	memorydb:DescribeSnapshots
	memorydb:DescribeSubnetGroups
	memorydb:DescribeUsers
	memorydb:FailoverShard
	memorydb:ListAllowedMultiRegionClusterUpdates
	memorydb:ListAllowedNodeTypeUpdates
	memorydb:PurchaseReservedNodesOffering
	memorydb:ResetParameterGroup
	memorydb:UpdateAcl
	memorydb:UpdateCluster
	memorydb:UpdateMultiRegionCluster
	memorydb:UpdateParameterGroup
	memorydb:UpdateSubnetGroup
	memorydb:UpdateUser

Service prefix	Actions
mgh	mgh:AssociateCreatedArtifact mgh:AssociateDiscoveredResource mgh:AssociateSourceResource mgh:CreateHomeRegionControl mgh:CreateProgressUpdateStream mgh:DeleteHomeRegionControl mgh:DeleteProgressUpdateStream mgh:DescribeApplicationState mgh:DescribeHomeRegionControls mgh:DescribeMigrationTask mgh:DisassociateCreatedArtifact mgh:DisassociateDiscoveredResource mgh:DisassociateSourceResource mgh:GetHomeRegion mgh:ImportMigrationTask mgh:ListApplicationStates mgh:ListCreatedArtifacts mgh:ListDiscoveredResources mgh:ListMigrationTaskUpdates mgh:ListMigrationTasks mgh:ListProgressUpdateStreams

Service prefix	Actions
	mgh:ListSourceResources mgh:NotifyApplicationState mgh:NotifyMigrationTaskState mgh:PutResourceAttributes

Service prefix	Actions
mgn	mgn:ArchiveApplication
	mgn:ArchiveWave
	mgn:AssociateApplications
	mgn:AssociateSourceServers
	mgn:ChangeServerLifeCycleState
	mgn:CreateApplication
	mgn:CreateConnector
	mgn:CreateLaunchConfigurationTemplate
	mgn:CreateNetworkMigrationDefinition
	mgn:CreateReplicationConfigurationTemplate
	mgn:CreateWave
	mgn:DeleteApplication
	mgn:DeleteConnector
	mgn:DeleteJob
	mgn:DeleteLaunchConfigurationTemplate
	mgn:DeleteNetworkMigrationDefinition
	mgn:DeleteReplicationConfigurationTemplate
	mgn:DeleteSourceServer
	mgn:DeleteVcenterClient
	mgn:DeleteWave
	mgn:DescribeJobLogItems

Service prefix	Actions
	mgn:DescribeJobs
	mgn:DescribeLaunchConfigurationTemplates
	mgn:DescribeReplicationConfigurationTemplates
	mgn:DescribeVcenterClients
	mgn:DisassociateApplications
	mgn:DisassociateSourceServers
	mgn:DisconnectFromService
	mgn:FinalizeCutover
	mgn:GetNetworkMigrationDefinition
	mgn:GetNetworkMigrationMapperSegmentConstruct
	mgn:GetReplicationConfiguration
	mgn:InitializeService
	mgn:ListConnectors
	mgn:ListExportErrors
	mgn:ListExports
	mgn:ListImportErrors
	mgn:ListImportFileEnrichments
	mgn:ListImports
	mgn:ListManagedAccounts
	mgn:ListNetworkMigrationAnalyses
	mgn:ListNetworkMigrationAnalysisResults

Service prefix	Actions
	mgn:ListNetworkMigrationCodeGenerationSegments
	mgn:ListNetworkMigrationCodeGenerations
	mgn:ListNetworkMigrationDefinitions
	mgn:ListNetworkMigrationDeployedStacks
	mgn:ListNetworkMigrationDeployments
	mgn:ListNetworkMigrationExecutions
	mgn:ListNetworkMigrationMapperSegmentConstructs
	mgn:ListNetworkMigrationMapperSegments
	mgn:ListNetworkMigrationMappingUpdates
	mgn:ListNetworkMigrationMappings
	mgn:ListSourceServerActions
	mgn:ListTemplateActions
	mgn:MarkAsArchived
	mgn:PauseReplication
	mgn:PutSourceServerAction
	mgn:PutTemplateAction
	mgn:RemoveSourceServerAction
	mgn:RemoveTemplateAction
	mgn:ResumeReplication
	mgn:RetryDataReplication
	mgn:StartCutover

Service prefix	Actions
	mgn:StartExport
	mgn:StartImport
	mgn:StartImportFileEnrichment
	mgn:StartNetworkMigrationCodeGeneration
	mgn:StartNetworkMigrationMapping
	mgn:StartNetworkMigrationMappingUpdate
	mgn:StartReplication
	mgn:StartTest
	mgn:StopReplication
	mgn:TerminateTargetInstances
	mgn:UnarchiveApplication
	mgn:UnarchiveWave
	mgn:UpdateApplication
	mgn:UpdateConnector
	mgn:UpdateLaunchConfigurationTemplate
	mgn:UpdateNetworkMigrationDefinition
	mgn:UpdateNetworkMigrationMapperSegment
	mgn:UpdateReplicationConfiguration
	mgn:UpdateReplicationConfigurationTemplate
	mgn:UpdateSourceServer
	mgn:UpdateSourceServerReplicationType

Service prefix	Actions
	mgn:UpdateWave

Service prefix	Actions
migrationhub-strategy	migrationhub-strategy:GetAntiPattern migrationhub-strategy:GetApplicationComponentDetails migrationhub-strategy:GetApplicationComponentStrategies migrationhub-strategy:GetAssessment migrationhub-strategy:GetImportFileTask migrationhub-strategy:GetLatestAssessmentId migrationhub-strategy:GetMessage migrationhub-strategy:GetPortfolioPreferences migrationhub-strategy:GetPortfolioSummary migrationhub-strategy:GetRecommendationReportDetails migrationhub-strategy:GetServerDetails migrationhub-strategy:GetServerStrategies migrationhub-strategy:ListAnalyzableServers migrationhub-strategy:ListAntiPatterns migrationhub-strategy:ListApplicationComponents migrationhub-strategy:ListCollectors migrationhub-strategy:ListImportFileTask migrationhub-strategy:ListJarArtifacts migrationhub-strategy:ListServers migrationhub-strategy:PutLogData migrationhub-strategy:PutMetricData

Service prefix	Actions
	migrationhub-strategy:PutPortfolioPreferences migrationhub-strategy:RegisterCollector migrationhub-strategy:SendMessage migrationhub-strategy:StartAssessment migrationhub-strategy:StartImportFileTask migrationhub-strategy:StartRecommendationReportGeneration migrationhub-strategy:StopAssessment migrationhub-strategy:UpdateApplicationComponentConfig migrationhub-strategy:UpdateCollectorConfiguration migrationhub-strategy:UpdateServerConfig

Service prefix	Actions
mobiletargeting	mobiletargeting:CreateApp
	mobiletargeting:CreateCampaign
	mobiletargeting:CreateEmailTemplate
	mobiletargeting:CreateExportJob
	mobiletargeting:CreateImportJob
	mobiletargeting:CreateInAppTemplate
	mobiletargeting:CreateJourney
	mobiletargeting:CreatePushTemplate
	mobiletargeting:CreateRecommenderConfiguration
	mobiletargeting:CreateSegment
	mobiletargeting:CreateSmsTemplate
	mobiletargeting:CreateVoiceTemplate
	mobiletargeting:DeleteAdmChannel
	mobiletargeting:DeleteApnsChannel
	mobiletargeting:DeleteApnsSandboxChannel
	mobiletargeting:DeleteApnsVoipChannel
	mobiletargeting:DeleteApnsVoipSandboxChannel
	mobiletargeting:DeleteApp
	mobiletargeting:DeleteBaiduChannel
	mobiletargeting:DeleteCampaign
	mobiletargeting:DeleteEmailChannel

Service prefix	Actions
	mobiletargeting:DeleteEmailTemplate
	mobiletargeting:DeleteEndpoint
	mobiletargeting:DeleteEventStream
	mobiletargeting:DeleteGcmChannel
	mobiletargeting:DeleteInAppTemplate
	mobiletargeting:DeleteJourney
	mobiletargeting:DeletePushTemplate
	mobiletargeting:DeleteRecommenderConfiguration
	mobiletargeting:DeleteSegment
	mobiletargeting:DeleteSmsChannel
	mobiletargeting:DeleteSmsTemplate
	mobiletargeting:DeleteUserEndpoints
	mobiletargeting:DeleteVoiceChannel
	mobiletargeting:DeleteVoiceTemplate
	mobiletargeting:GetAdmChannel
	mobiletargeting:GetApnsChannel
	mobiletargeting:GetApnsSandboxChannel
	mobiletargeting:GetApnsVoipChannel
	mobiletargeting:GetApnsVoipSandboxChannel
	mobiletargeting:GetApp
	mobiletargeting:GetApplicationDateRangeKpi

Service prefix	Actions
	mobiletargeting:GetApplicationSettings
	mobiletargeting:GetApps
	mobiletargeting:GetBaiduChannel
	mobiletargeting:GetCampaign
	mobiletargeting:GetCampaignActivities
	mobiletargeting:GetCampaignDateRangeKpi
	mobiletargeting:GetCampaignVersion
	mobiletargeting:GetCampaignVersions
	mobiletargeting:GetCampaigns
	mobiletargeting:GetChannels
	mobiletargeting:GetEmailChannel
	mobiletargeting:GetEmailTemplate
	mobiletargeting:GetEndpoint
	mobiletargeting:GetEventStream
	mobiletargeting:GetExportJob
	mobiletargeting:GetExportJobs
	mobiletargeting:GetGcmChannel
	mobiletargeting:GetImportJob
	mobiletargeting:GetImportJobs
	mobiletargeting:GetInAppMessages
	mobiletargeting:GetInAppTemplate

Service prefix	Actions
	mobiletargeting:GetJourney
	mobiletargeting:GetJourneyDateRangeKpi
	mobiletargeting:GetJourneyExecutionActivityMetrics
	mobiletargeting:GetJourneyExecutionMetrics
	mobiletargeting:GetJourneyRunExecutionActivityMetrics
	mobiletargeting:GetJourneyRunExecutionMetrics
	mobiletargeting:GetJourneyRuns
	mobiletargeting:GetPushTemplate
	mobiletargeting:GetRecommenderConfiguration
	mobiletargeting:GetRecommenderConfigurations
	mobiletargeting:GetSegment
	mobiletargeting:GetSegmentExportJobs
	mobiletargeting:GetSegmentImportJobs
	mobiletargeting:GetSegmentVersion
	mobiletargeting:GetSegmentVersions
	mobiletargeting:GetSegments
	mobiletargeting:GetSmsChannel
	mobiletargeting:GetSmsTemplate
	mobiletargeting:GetUserEndpoints
	mobiletargeting:GetVoiceChannel
	mobiletargeting:GetVoiceTemplate

Service prefix	Actions
	mobiletargeting:ListJourneys
	mobiletargeting:ListTemplateVersions
	mobiletargeting:ListTemplates
	mobiletargeting:PhoneNumberValidate
	mobiletargeting:PutEventStream
	mobiletargeting:RemoveAttributes
	mobiletargeting:UpdateAdmChannel
	mobiletargeting:UpdateApnsChannel
	mobiletargeting:UpdateApnsSandboxChannel
	mobiletargeting:UpdateApnsVoipChannel
	mobiletargeting:UpdateApnsVoipSandboxChannel
	mobiletargeting:UpdateApplicationSettings
	mobiletargeting:UpdateBaiduChannel
	mobiletargeting:UpdateCampaign
	mobiletargeting:UpdateEmailChannel
	mobiletargeting:UpdateEmailTemplate
	mobiletargeting:UpdateEndpoint
	mobiletargeting:UpdateEndpointsBatch
	mobiletargeting:UpdateGcmChannel
	mobiletargeting:UpdateInAppTemplate
	mobiletargeting:UpdateJourney

Service prefix	Actions
	mobiletargeting:UpdateJourneyState
	mobiletargeting:UpdatePushTemplate
	mobiletargeting:UpdateRecommenderConfiguration
	mobiletargeting:UpdateSegment
	mobiletargeting:UpdateSmsChannel
	mobiletargeting:UpdateSmsTemplate
	mobiletargeting:UpdateTemplateActiveVersion
	mobiletargeting:UpdateVoiceChannel
	mobiletargeting:UpdateVoiceTemplate
	mobiletargeting:VerifyOTPMessage

Service prefix	Actions
mq	mq:CreateBroker mq:CreateConfiguration mq:CreateUser mq:DeleteBroker mq:DeleteConfiguration mq:DeleteUser mq:DescribeBroker mq:DescribeBrokerEngineTypes mq:DescribeBrokerInstanceOptions mq:DescribeConfiguration mq:DescribeConfigurationRevision mq:DescribeSharedResources mq:DescribeUser mq:ListBrokers mq:ListConfigurationRevisions mq:ListConfigurations mq:ListUsers mq:Promote mq:RebootBroker mq:UpdateBroker mq:UpdateConfiguration

Service prefix	Actions
	mq:UpdateUser

Service prefix	Actions
networkmanager	networkmanager:AcceptAttachment
	networkmanager:AssociateConnectPeer
	networkmanager:AssociateCustomerGateway
	networkmanager:AssociateLink
	networkmanager:AssociateTransitGatewayConnectPeer
	networkmanager:CreateConnectAttachment
	networkmanager:CreateConnectPeer
	networkmanager:CreateConnection
	networkmanager:CreateCoreNetwork
	networkmanager:CreateDevice
	networkmanager:CreateDirectConnectGatewayAttachment
	networkmanager:CreateGlobalNetwork
	networkmanager:CreateLink
	networkmanager:CreateSite
	networkmanager:CreateSiteToSiteVpnAttachment
	networkmanager:CreateTransitGatewayPeering
	networkmanager:CreateTransitGatewayRouteTableAttachment
	networkmanager:CreateVpcAttachment
	networkmanager:DeleteAttachment
	networkmanager:DeleteConnectPeer
	networkmanager:DeleteConnection

Service prefix	Actions
	networkmanager:DeleteCoreNetwork
	networkmanager:DeleteCoreNetworkPolicyVersion
	networkmanager:DeleteDevice
	networkmanager:DeleteGlobalNetwork
	networkmanager:DeleteLink
	networkmanager:DeletePeering
	networkmanager:DeleteResourcePolicy
	networkmanager:DeleteSite
	networkmanager:DeregisterTransitGateway
	networkmanager:DescribeGlobalNetworks
	networkmanager:DisassociateConnectPeer
	networkmanager:DisassociateCustomerGateway
	networkmanager:DisassociateLink
	networkmanager:DisassociateTransitGatewayConnectPeer
	networkmanager:ExecuteCoreNetworkChangeSet
	networkmanager:GetConnectAttachment
	networkmanager:GetConnectPeer
	networkmanager:GetConnectPeerAssociations
	networkmanager:GetConnections
	networkmanager:GetCoreNetwork
	networkmanager:GetCoreNetworkChangeEvents

Service prefix	Actions
	networkmanager:GetCoreNetworkChangeSet
	networkmanager:GetCoreNetworkPolicy
	networkmanager:GetCustomerGatewayAssociations
	networkmanager:GetDevices
	networkmanager:GetLinkAssociations
	networkmanager:GetLinks
	networkmanager:GetNetworkResourceCounts
	networkmanager:GetNetworkResourceRelationships
	networkmanager:GetNetworkResources
	networkmanager:GetNetworkRoutes
	networkmanager:GetNetworkTelemetry
	networkmanager:GetResourcePolicy
	networkmanager:GetRouteAnalysis
	networkmanager:GetSiteToSiteVpnAttachment
	networkmanager:GetSites
	networkmanager:GetTransitGatewayConnectPeerAssociations
	networkmanager:GetTransitGatewayPeering
	networkmanager:GetTransitGatewayRegistrations
	networkmanager:GetTransitGatewayRouteTableAttachment
	networkmanager:GetVpcAttachment
	networkmanager:ListAttachmentRoutingPolicyAssociations

Service prefix	Actions
	networkmanager:ListAttachments
	networkmanager:ListConnectPeers
	networkmanager:ListCoreNetworkPolicyVersions
	networkmanager:ListCoreNetworkPrefixListAssociations
	networkmanager:ListCoreNetworkRoutingInformation
	networkmanager:ListCoreNetworks
	networkmanager:ListOrganizationServiceAccessStatus
	networkmanager:ListPeerings
	networkmanager:PutAttachmentRoutingPolicyLabel
	networkmanager:PutCoreNetworkPolicy
	networkmanager:PutResourcePolicy
	networkmanager:RegisterTransitGateway
	networkmanager:RejectAttachment
	networkmanager:RemoveAttachmentRoutingPolicyLabel
	networkmanager:RestoreCoreNetworkPolicyVersion
	networkmanager:StartOrganizationServiceAccessUpdate
	networkmanager:StartRouteAnalysis
	networkmanager:UpdateConnection
	networkmanager:UpdateCoreNetwork
	networkmanager:UpdateDevice
	networkmanager:UpdateDirectConnectGatewayAttachment

Service prefix	Actions
	networkmanager:UpdateGlobalNetwork networkmanager:UpdateLink networkmanager:UpdateNetworkResourceMetadata networkmanager:UpdateSite networkmanager:UpdateVpcAttachment

Service prefix	Actions
nimble	nimble:AcceptEulas nimble:CreateLaunchProfile nimble:CreateStreamingImage nimble:CreateStreamingSession nimble:CreateStreamingSessionStream nimble:CreateStudio nimble:CreateStudioComponent nimble:DeleteLaunchProfile nimble:DeleteLaunchProfileMember nimble:DeleteStreamingImage nimble:DeleteStreamingSession nimble:DeleteStudio nimble:DeleteStudioComponent nimble:DeleteStudioMember nimble:GetEula nimble:GetLaunchProfileDetails nimble:GetStreamingImage nimble:GetStreamingSession nimble:GetStreamingSessionBackup nimble:GetStreamingSessionStream nimble:GetStudio

Service prefix	Actions
	nimble:GetStudioComponent nimble:GetStudioMember nimble:ListEulas nimble:ListLaunchProfileMembers nimble:ListLaunchProfiles nimble:ListStreamingImages nimble:ListStreamingSessionBackups nimble:ListStreamingSessions nimble:ListStudioComponents nimble:ListStudioMembers nimble:ListStudios nimble:PutLaunchProfileMembers nimble:PutStudioMembers nimble:StartStreamingSession nimble:StartStudioSSOConfigurationRepair nimble:StopStreamingSession nimble:UpdateLaunchProfile nimble:UpdateLaunchProfileMember nimble:UpdateStreamingImage nimble:UpdateStudio nimble:UpdateStudioComponent

Service prefix	Actions
omics	omics:AbortMultipartReadSetUpload
	omics:AcceptShare
	omics:BatchDeleteReadSet
	omics:CancelAnnotationImportJob
	omics:CancelRun
	omics:CancelRunBatch
	omics:CancelVariantImportJob
	omics:CompleteMultipartReadSetUpload
	omics:CreateAnnotationStore
	omics:CreateAnnotationStoreVersion
	omics:CreateConfiguration
	omics:CreateMultipartReadSetUpload
	omics:CreateReferenceStore
	omics:CreateRunGroup
	omics:CreateSequenceStore
	omics:CreateShare
	omics:CreateVariantStore
	omics:CreateWorkflow
	omics:CreateWorkflowVersion
	omics:DeleteAnnotationStore
	omics:DeleteAnnotationStoreVersions

Service prefix	Actions
	omics:DeleteBatch
	omics:DeleteConfiguration
	omics:DeleteReference
	omics:DeleteReferenceStore
	omics:DeleteRun
	omics:DeleteRunBatch
	omics:DeleteRunGroup
	omics:DeleteSequenceStore
	omics:DeleteShare
	omics:DeleteVariantStore
	omics:DeleteWorkflow
	omics:DeleteWorkflowVersion
	omics:GetAnnotationImportJob
	omics:GetAnnotationStore
	omics:GetAnnotationStoreVersion
	omics:GetBatch
	omics:GetConfiguration
	omics:GetReadSet
	omics:GetReadSetActivationJob
	omics:GetReadSetExportJob
	omics:GetReadSetImportJob

Service prefix	Actions
	omics:GetReadSetMetadata
	omics:GetReference
	omics:GetReferenceImportJob
	omics:GetReferenceMetadata
	omics:GetReferenceStore
	omics:GetRun
	omics:GetRunGroup
	omics:GetRunTask
	omics:GetSequenceStore
	omics:GetShare
	omics:GetVariantImportJob
	omics:GetVariantStore
	omics:GetWorkflow
	omics:GetWorkflowVersion
	omics:ListAnnotationImportJobs
	omics:ListAnnotationStoreVersions
	omics:ListAnnotationStores
	omics:ListBatch
	omics:ListConfigurations
	omics:ListMultipartReadSetUploads
	omics:ListReadSetActivationJobs

Service prefix	Actions
	omics:ListReadSetExportJobs
	omics:ListReadSetImportJobs
	omics:ListReadSetUploadParts
	omics:ListReadSets
	omics:ListReferenceImportJobs
	omics:ListReferenceStores
	omics:ListReferences
	omics:ListRunGroups
	omics:ListRunTasks
	omics:ListRuns
	omics:ListRunsInBatch
	omics:ListSequenceStores
	omics:ListShares
	omics:ListVariantImportJobs
	omics:ListVariantStores
	omics:ListWorkflowVersions
	omics:ListWorkflows
	omics:StartAnnotationImportJob
	omics:StartReadSetActivationJob
	omics:StartReadSetExportJob
	omics:StartReadSetImportJob

Service prefix	Actions
	omics:StartReferenceImportJob omics:StartRun omics:StartRunBatch omics:StartVariantImportJob omics:UpdateAnnotationStore omics:UpdateAnnotationStoreVersion omics:UpdateRunGroup omics:UpdateVariantStore omics:UpdateWorkflow omics:UpdateWorkflowVersion omics:UploadReadSetPart

Service prefix	Actions
opsworks	opsworks:AssignInstance opsworks:AssignVolume opsworks:AssociateElasticIp opsworks:AttachElasticLoadBalancer opsworks:CloneStack opsworks:CreateApp opsworks:CreateDeployment opsworks:CreateInstance opsworks:CreateLayer opsworks:CreateStack opsworks:CreateUserProfile opsworks:DeleteApp opsworks:DeleteInstance opsworks:DeleteLayer opsworks:DeleteStack opsworks:DeleteUserProfile opsworks:DeregisterEcsCluster opsworks:DeregisterElasticIp opsworks:DeregisterInstance opsworks:DeregisterRdsDbInstance opsworks:DeregisterVolume

Service prefix	Actions
	opsworks:DescribeAgentVersions opsworks:DescribeApps opsworks:DescribeCommands opsworks:DescribeDeployments opsworks:DescribeEcsClusters opsworks:DescribeElasticIps opsworks:DescribeElasticLoadBalancers opsworks:DescribeInstances opsworks:DescribeLayers opsworks:DescribeLoadBasedAutoScaling opsworks:DescribeMyUserProfile opsworks:DescribeOperatingSystems opsworks:DescribePermissions opsworks:DescribeRaidArrays opsworks:DescribeRdsDbInstances opsworks:DescribeServiceErrors opsworks:DescribeStackProvisioningParameters opsworks:DescribeStackSummary opsworks:DescribeStacks opsworks:DescribeTimeBasedAutoScaling opsworks:DescribeUserProfiles

Service prefix	Actions
	opsworks:DescribeVolumes opsworks:DetachElasticLoadBalancer opsworks:DisassociateElasticIp opsworks:GetHostnameSuggestion opsworks:GrantAccess opsworks:RebootInstance opsworks:RegisterEcsCluster opsworks:RegisterElasticIp opsworks:RegisterInstance opsworks:RegisterRdsDbInstance opsworks:RegisterVolume opsworks:SetLoadBasedAutoScaling opsworks:SetPermission opsworks:SetTimeBasedAutoScaling opsworks:StartInstance opsworks:StartStack opsworks:StopInstance opsworks:StopStack opsworks:UnassignInstance opsworks:UnassignVolume opsworks:UpdateApp

Service prefix	Actions
	opsworks:UpdateElasticIp opsworks:UpdateInstance opsworks:UpdateLayer opsworks:UpdateMyUserProfile opsworks:UpdateRdsDbInstance opsworks:UpdateStack opsworks:UpdateUserProfile opsworks:UpdateVolume

Service prefix	Actions
opsworks-cm	opsworks-cm:AssociateNode
	opsworks-cm:CreateBackup
	opsworks-cm:CreateServer
	opsworks-cm:DeleteBackup
	opsworks-cm:DeleteServer
	opsworks-cm:DescribeAccountAttributes
	opsworks-cm:DescribeBackups
	opsworks-cm:DescribeEvents
	opsworks-cm:DescribeNodeAssociationStatus
	opsworks-cm:DescribeServers
	opsworks-cm:DisassociateNode
	opsworks-cm:ExportServerEngineAttribute
	opsworks-cm:RestoreServer
	opsworks-cm:StartMaintenance
	opsworks-cm:UpdateServer
	opsworks-cm:UpdateServerEngineAttributes

Service prefix	Actions
organizations	organizations:AcceptHandshake organizations:AttachPolicy organizations:CancelHandshake organizations:CloseAccount organizations:CreateAccount organizations:CreateGovCloudAccount organizations:CreateOrganization organizations:CreateOrganizationalUnit organizations:CreatePolicy organizations:DeclineHandshake organizations:DeleteOrganization organizations:DeleteOrganizationalUnit organizations:DeletePolicy organizations:DeleteResourcePolicy organizations:DeregisterDelegatedAdministrator organizations:DescribeAccount organizations:DescribeCreateAccountStatus organizations:DescribeEffectivePolicy organizations:DescribeHandshake organizations:DescribeOrganization organizations:DescribeOrganizationalUnit

Service prefix	Actions
	organizations:DescribePolicy organizations:DescribeResourcePolicy organizations:DescribeResponsibilityTransfer organizations:DetachPolicy organizations:DisableAWSServiceAccess organizations:DisablePolicyType organizations:EnableAWSServiceAccess organizations:EnableAllFeatures organizations:EnablePolicyType organizations:InviteAccountToOrganization organizations:LeaveOrganization organizations:ListAWSServiceAccessForOrganization organizations:ListAccounts organizations:ListAccountsForParent organizations:ListAccountsWithInvalidEffectivePolicy organizations:ListChildren organizations:ListCreateAccountStatus organizations:ListDelegatedAdministrators organizations:ListDelegatedServicesForAccount organizations:ListHandshakesForAccount organizations:ListHandshakesForOrganization

Service prefix	Actions
	organizations:ListInboundResponsibilityTransfers organizations:ListOrganizationalUnitsForParent organizations:ListOutboundResponsibilityTransfers organizations:ListParents organizations:ListPolicies organizations:ListPoliciesForTarget organizations:ListRoots organizations:ListTargetsForPolicy organizations:MoveAccount organizations:PutResourcePolicy organizations:RegisterDelegatedAdministrator organizations:RemoveAccountFromOrganization organizations:TerminateResponsibilityTransfer organizations:UpdateOrganizationalUnit organizations:UpdatePolicy organizations:UpdateResponsibilityTransfer

Service prefix	Actions
outposts	outposts:CancelCapacityTask
	outposts:CancelOrder
	outposts:CreateOrder
	outposts:CreateOutpost
	outposts:CreatePrivateConnectivityConfig
	outposts:CreateQuote
	outposts:CreateRenewal
	outposts:CreateSite
	outposts:DeleteOutpost
	outposts:DeleteQuote
	outposts:DeleteSite
	outposts:GetCapacityTask
	outposts:GetCatalogItem
	outposts:GetConnection
	outposts:GetOrder
	outposts:GetOutpost
	outposts:GetOutpostBillingInformation
	outposts:GetOutpostInstanceTypes
	outposts:GetOutpostSupportedInstanceTypes
	outposts:GetPrivateConnectivityConfig
	outposts:GetQuote

Service prefix	Actions
	outposts:GetRenewalPricing
	outposts:GetSite
	outposts:GetSiteAddress
	outposts:ListAssetInstances
	outposts:ListAssets
	outposts:ListBlockingInstancesForCapacityTask
	outposts:ListCapacityTasks
	outposts:ListCatalogItems
	outposts:ListOrderableInstanceTypes
	outposts:ListOrders
	outposts:ListOutposts
	outposts:ListQuotes
	outposts:ListSites
	outposts:StartCapacityTask
	outposts:StartConnection
	outposts:UpdateOutpost
	outposts:UpdateQuote
	outposts:UpdateSite
	outposts:UpdateSiteAddress
	outposts:UpdateSiteRackPhysicalProperties

Service prefix	Actions
pi	pi:CreatePerformanceAnalysisReport pi:DeletePerformanceAnalysisReport pi:DescribeDimensionKeys pi:GetDimensionKeyDetails pi:GetPerformanceAnalysisReport pi:GetResourceMetadata pi:GetResourceMetrics pi:ListAvailableResourceDimensions pi:ListAvailableResourceMetrics pi:ListPerformanceAnalysisReports
pipes	pipes:CreatePipe pipes:DeletePipe pipes:DescribePipe pipes:ListPipes pipes:StartPipe pipes:StopPipe pipes:UpdatePipe

Service prefix	Actions
polly	polly:DeleteLexicon polly:DescribeVoices polly:GetLexicon polly:GetSpeechSynthesisTask polly:ListLexicons polly:ListSpeechSynthesisTasks polly:PutLexicon polly:StartSpeechSynthesisTask polly:SynthesizeSpeech

Service prefix	Actions
profile	profile:AddProfileKey
	profile:BatchGetCalculatedAttributeForProfile
	profile:BatchGetProfile
	profile:CreateCalculatedAttributeDefinition
	profile:CreateDomain
	profile:CreateEventStream
	profile:CreateProfile
	profile:CreateRecommender
	profile:CreateRecommenderFilter
	profile:CreateRecommenderSchema
	profile:CreateSegmentDefinition
	profile:CreateSegmentEstimate
	profile:CreateSegmentSnapshot
	profile:CreateUploadJob
	profile:DeleteCalculatedAttributeDefinition
	profile:DeleteDomain
	profile:DeleteDomainObjectType
	profile:DeleteEventStream
	profile:DeleteIntegration
	profile:DeleteProfile
	profile:DeleteProfileKey

Service prefix	Actions
	profile:DeleteProfileObject
	profile:DeleteProfileObjectType
	profile:DeleteRecommender
	profile:DeleteRecommenderFilter
	profile:DeleteRecommenderSchema
	profile:DeleteSegmentDefinition
	profile:DeleteWorkflow
	profile:DetectProfileObjectType
	profile:GetAutoMergingPreview
	profile:GetCalculatedAttributeDefinition
	profile:GetCalculatedAttributeForProfile
	profile:GetDomain
	profile:GetDomainObjectType
	profile:GetEventStream
	profile:GetIdentityResolutionJob
	profile:GetIntegration
	profile:GetMatches
	profile:GetObjectTypeAttributeStatistics
	profile:GetProfileObjectType
	profile:GetProfileObjectTypeTemplate
	profile:GetProfileRecommendations

Service prefix	Actions
	profile:GetRecommender
	profile:GetRecommenderFilter
	profile:GetRecommenderSchema
	profile:GetSegmentDefinition
	profile:GetSegmentEstimate
	profile:GetSegmentMembership
	profile:GetSegmentSnapshot
	profile:GetSimilarProfiles
	profile:GetUploadJob
	profile:GetUploadJobPath
	profile:GetWorkflow
	profile:GetWorkflowSteps
	profile:ListAccountIntegrations
	profile:ListCalculatedAttributeDefinitions
	profile:ListCalculatedAttributesForProfile
	profile:ListDomainLayouts
	profile:ListDomainObjectTypes
	profile:ListDomains
	profile:ListEventStreams
	profile:ListIdentityResolutionJobs
	profile:ListIntegrations

Service prefix	Actions
	profile:ListObjectTypeAttributeValues
	profile:ListObjectTypeAttributes
	profile:ListProfileAttributeValues
	profile:ListProfileObjectTypeTemplates
	profile:ListProfileObjectTypes
	profile:ListProfileObjects
	profile:ListRecommenderFilters
	profile:ListRecommenderRecipes
	profile:ListRecommenderSchemas
	profile:ListRecommenders
	profile:ListRuleBasedMatches
	profile:ListSegmentDefinitions
	profile:ListUploadJobs
	profile:ListWorkflows
	profile:MergeProfiles
	profile:PutDomainObjectType
	profile:PutIntegration
	profile:PutProfileObject
	profile:PutProfileObjectType
	profile:SearchProfiles
	profile:StartRecommender

Service prefix	Actions
	profile:StartUploadJob profile:StopRecommender profile:StopUploadJob profile:UpdateCalculatedAttributeDefinition profile:UpdateDomain profile:UpdateProfile profile:UpdateRecommender

Service prefix	Actions
qldb	qldb:CancelJournalKinesisStream qldb:CreateLedger qldb:DeleteLedger qldb:DescribeJournalKinesisStream qldb:DescribeJournalS3Export qldb:DescribeLedger qldb:ExportJournalToS3 qldb:GetBlock qldb:GetDigest qldb:GetRevision qldb:ListJournalKinesisStreamsForLedger qldb:ListJournalS3Exports qldb:ListJournalS3ExportsForLedger qldb:ListLedgers qldb:StreamJournalToKinesis qldb:UpdateLedger qldb:UpdateLedgerPermissionsMode

Service prefix	Actions
ram	ram:AcceptResourceShareInvitation
	ram:AssociateResourceShare
	ram:AssociateResourceSharePermission
	ram:CreatePermission
	ram:CreatePermissionVersion
	ram:CreateResourceShare
	ram:DeletePermission
	ram:DeletePermissionVersion
	ram:DeleteResourceShare
	ram:DisassociateResourceShare
	ram:DisassociateResourceSharePermission
	ram:EnableSharingWithAwsOrganization
	ram:GetPermission
	ram:GetResourcePolicies
	ram:GetResourceShareAssociations
	ram:GetResourceShareInvitations
	ram:GetResourceShares
	ram:ListPendingInvitationResources
	ram:ListPermissionAssociations
	ram:ListPermissionVersions
	ram:ListPermissions

Service prefix	Actions
	ram:ListPrincipals ram:ListReplacePermissionAssociationsWork ram:ListResourceSharePermissions ram:ListResourceTypes ram:ListResources ram:ListSourceAssociations ram:PromotePermissionCreatedFromPolicy ram:PromoteResourceShareCreatedFromPolicy ram:RejectResourceShareInvitation ram:ReplacePermissionAssociations ram:SetDefaultPermissionVersion ram:UpdateResourceShare
rbin	rbin:CreateRule rbin:DeleteRule rbin:GetRule rbin:ListRules rbin:LockRule rbin:UnlockRule rbin:UpdateRule

Service prefix	Actions
rds	rds:AddRoleToDBCluster
	rds:AddRoleToDBInstance
	rds:AddSourceIdentifierToSubscription
	rds:ApplyPendingMaintenanceAction
	rds:AuthorizeDBSecurityGroupIngress
	rds:BacktrackDBCluster
	rds:CancelExportTask
	rds:CopyDBClusterParameterGroup
	rds:CopyDBClusterSnapshot
	rds:CopyDBParameterGroup
	rds:CopyDBSnapshot
	rds:CopyOptionGroup
	rds:CreateCustomDBEngineVersion
	rds:CreateDBClusterParameterGroup
	rds:CreateDBParameterGroup
	rds:CreateDBProxy
	rds:CreateDBProxyEndpoint
	rds:CreateDBSecurityGroup
	rds:CreateDBSubnetGroup
	rds:CreateEventSubscription
	rds:CreateGlobalCluster

Service prefix	Actions
	rds:CreateOptionGroup
	rds:DeleteBlueGreenDeployment
	rds:DeleteDBClusterAutomatedBackup
	rds:DeleteDBClusterParameterGroup
	rds:DeleteDBClusterSnapshot
	rds:DeleteDBInstanceAutomatedBackup
	rds:DeleteDBParameterGroup
	rds:DeleteDBProxy
	rds:DeleteDBProxyEndpoint
	rds:DeleteDBSecurityGroup
	rds:DeleteDBSnapshot
	rds:DeleteDBSubnetGroup
	rds:DeleteEventSubscription
	rds:DeleteGlobalCluster
	rds:DeleteOptionGroup
	rds:DeregisterDBProxyTargets
	rds:DescribeAccountAttributes
	rds:DescribeBlueGreenDeployments
	rds:DescribeCertificates
	rds:DescribeDBClusterAutomatedBackups
	rds:DescribeDBClusterBacktracks

Service prefix	Actions
	rds:DescribeDBClusterEndpoints
	rds:DescribeDBClusterParameterGroups
	rds:DescribeDBClusterParameters
	rds:DescribeDBClusterSnapshotAttributes
	rds:DescribeDBClusterSnapshots
	rds:DescribeDBClusters
	rds:DescribeDBEngineVersions
	rds:DescribeDBInstanceAutomatedBackups
	rds:DescribeDBInstances
	rds:DescribeDBLogFiles
	rds:DescribeDBMajorEngineVersions
	rds:DescribeDBParameterGroups
	rds:DescribeDBParameters
	rds:DescribeDBProxies
	rds:DescribeDBProxyEndpoints
	rds:DescribeDBProxyTargetGroups
	rds:DescribeDBProxyTargets
	rds:DescribeDBRecommendations
	rds:DescribeDBSecurityGroups
	rds:DescribeDBSnapshotAttributes
	rds:DescribeDBSnapshotTenantDatabases

Service prefix	Actions
	rds:DescribeDBSnapshots
	rds:DescribeDBSubnetGroups
	rds:DescribeEngineDefaultClusterParameters
	rds:DescribeEngineDefaultParameters
	rds:DescribeEventCategories
	rds:DescribeEventSubscriptions
	rds:DescribeEvents
	rds:DescribeExportTasks
	rds:DescribeGlobalClusters
	rds:DescribeIntegrations
	rds:DescribeOptionGroupOptions
	rds:DescribeOptionGroups
	rds:DescribeOrderableDBInstanceOptions
	rds:DescribePendingMaintenanceActions
	rds:DescribeReservedDBInstances
	rds:DescribeReservedDBInstancesOfferings
	rds:DescribeSourceRegions
	rds:DescribeTenantDatabases
	rds:DescribeValidDBInstanceModifications
	rds:DownloadCompleteDBLogFile
	rds:DownloadDBLogFilePortion

Service prefix	Actions
	rds:FailoverDBCluster
	rds:FailoverGlobalCluster
	rds:ModifyActivityStream
	rds:ModifyCertificates
	rds:ModifyCurrentDBClusterCapacity
	rds:ModifyDBClusterEndpoint
	rds:ModifyDBClusterParameterGroup
	rds:ModifyDBClusterSnapshotAttribute
	rds:ModifyDBParameterGroup
	rds:ModifyDBProxy
	rds:ModifyDBProxyEndpoint
	rds:ModifyDBProxyTargetGroup
	rds:ModifyDBRecommendation
	rds:ModifyDBSnapshot
	rds:ModifyDBSnapshotAttribute
	rds:ModifyDBSubnetGroup
	rds:ModifyEventSubscription
	rds:ModifyGlobalCluster
	rds:ModifyOptionGroup
	rds:ModifyTenantDatabase
	rds:PurchaseReservedDBInstancesOffering

Service prefix	Actions
	rds:RebootDBCluster
	rds:RegisterDBProxyTargets
	rds:RemoveFromGlobalCluster
	rds:RemoveRoleFromDBCluster
	rds:RemoveRoleFromDBInstance
	rds:RemoveSourceIdentifierFromSubscription
	rds:ResetDBClusterParameterGroup
	rds:ResetDBParameterGroup
	rds:RestoreDBClusterFromS3
	rds:RestoreDBClusterFromSnapshot
	rds:RestoreDBClusterToPointInTime
	rds:RestoreDBInstanceFromDBSnapshot
	rds:RestoreDBInstanceFromS3
	rds:RestoreDBInstanceToPointInTime
	rds:RevokeDBSecurityGroupIngress
	rds:StartActivityStream
	rds:StartDBCluster
	rds:StartDBInstance
	rds:StartDBInstanceAutomatedBackupsReplication
	rds:StartExportTask
	rds:StopActivityStream

Service prefix	Actions
	rds:StopDBCluster rds:StopDBInstance rds:StopDBInstanceAutomatedBackupsReplication rds:SwitchoverBlueGreenDeployment rds:SwitchoverGlobalCluster rds:SwitchoverReadReplica

Service prefix	Actions
redshift	redshift:AcceptReservedNodeExchange redshift:AddPartner redshift:AssociateDataShareConsumer redshift:AuthorizeClusterSecurityGroupIngress redshift:AuthorizeDataShare redshift:AuthorizeEndpointAccess redshift:AuthorizeSnapshotAccess redshift:BatchDeleteClusterSnapshots redshift:BatchModifyClusterSnapshots redshift:CancelResize redshift:CopyClusterSnapshot redshift:CreateAuthenticationProfile redshift:CreateCluster redshift:CreateClusterParameterGroup redshift:CreateClusterSecurityGroup redshift:CreateClusterSnapshot redshift:CreateClusterSubnetGroup redshift:CreateCustomDomainAssociation redshift:CreateEndpointAccess redshift:CreateEventSubscription redshift:CreateHsmClientCertificate

Service prefix	Actions
	redshift:CreateHsmConfiguration redshift:CreateIntegration redshift:CreateRedshiftIdcApplication redshift:CreateScheduledAction redshift:CreateSnapshotCopyGrant redshift:CreateSnapshotSchedule redshift:CreateUsageLimit redshift:DeauthorizeDataShare redshift:DeleteAuthenticationProfile redshift:DeleteCluster redshift:DeleteClusterParameterGroup redshift:DeleteClusterSecurityGroup redshift:DeleteClusterSnapshot redshift:DeleteClusterSubnetGroup redshift:DeleteCustomDomainAssociation redshift:DeleteEndpointAccess redshift:DeleteEventSubscription redshift:DeleteHsmClientCertificate redshift:DeleteHsmConfiguration redshift:DeletePartner redshift:DeleteRedshiftIdcApplication

Service prefix	Actions
	redshift:DeleteResourcePolicy redshift:DeleteScheduledAction redshift:DeleteSnapshotCopyGrant redshift:DeleteSnapshotSchedule redshift:DeleteUsageLimit redshift:DeregisterNamespace redshift:DescribeAccountAttributes redshift:DescribeAuthenticationProfiles redshift:DescribeClusterDbRevisions redshift:DescribeClusterParameterGroups redshift:DescribeClusterParameters redshift:DescribeClusterSecurityGroups redshift:DescribeClusterSnapshots redshift:DescribeClusterSubnetGroups redshift:DescribeClusterTracks redshift:DescribeClusterVersions redshift:DescribeClusters redshift:DescribeCustomDomainAssociations redshift:DescribeDataShares redshift:DescribeDataSharesForConsumer redshift:DescribeDataSharesForProducer

Service prefix	Actions
	redshift:DescribeDefaultClusterParameters redshift:DescribeEndpointAccess redshift:DescribeEndpointAuthorization redshift:DescribeEventCategories redshift:DescribeEventSubscriptions redshift:DescribeEvents redshift:DescribeHsmClientCertificates redshift:DescribeHsmConfigurations redshift:DescribeInboundIntegrations redshift:DescribeIntegrations redshift:DescribeLoggingStatus redshift:DescribeNodeConfigurationOptions redshift:DescribeOrderableClusterOptions redshift:DescribePartners redshift:DescribeRedshiftIdcApplications redshift:DescribeReservedNodeExchangeStatus redshift:DescribeReservedNodeOfferings redshift:DescribeReservedNodes redshift:DescribeResize redshift:DescribeScheduledActions redshift:DescribeSnapshotCopyGrants

Service prefix	Actions
	redshift:DescribeSnapshotSchedules redshift:DescribeStorage redshift:DescribeTableRestoreStatus redshift:DescribeUsageLimits redshift:DisableLogging redshift:DisableSnapshotCopy redshift:DisassociateDataShareConsumer redshift:EnableLogging redshift:EnableSnapshotCopy redshift:FailoverPrimaryCompute redshift:GetClusterCredentials redshift:GetClusterCredentialsWithIAM redshift:GetIdentityCenterAuthToken redshift:GetReservedNodeExchangeConfigurationOptions redshift:GetReservedNodeExchangeOfferings redshift:GetResourcePolicy redshift:ListRecommendations redshift:ModifyAquaConfiguration redshift:ModifyAuthenticationProfile redshift:ModifyCluster redshift:ModifyClusterDbRevision

Service prefix	Actions
	redshift:ModifyClusterIamRoles redshift:ModifyClusterMaintenance redshift:ModifyClusterParameterGroup redshift:ModifyClusterSnapshot redshift:ModifyClusterSnapshotSchedule redshift:ModifyClusterSubnetGroup redshift:ModifyCustomDomainAssociation redshift:ModifyEndpointAccess redshift:ModifyEventSubscription redshift:ModifyRedshiftIdcApplication redshift:ModifyScheduledAction redshift:ModifySnapshotCopyRetentionPeriod redshift:ModifySnapshotSchedule redshift:ModifyUsageLimit redshift:PauseCluster redshift:PurchaseReservedNodeOffering redshift:PutResourcePolicy redshift:RebootCluster redshift:RegisterNamespace redshift:RejectDataShare redshift:ResetClusterParameterGroup

Service prefix	Actions
	redshift:ResizeCluster redshift:RestoreFromClusterSnapshot redshift:RestoreTableFromClusterSnapshot redshift:ResumeCluster redshift:RevokeClusterSecurityGroupIngress redshift:RevokeEndpointAccess redshift:RevokeSnapshotAccess redshift:RotateEncryptionKey redshift:UpdatePartnerStatus
redshift-data	redshift-data:BatchExecuteStatement redshift-data:CancelStatement redshift-data:DescribeStatement redshift-data:DescribeTable redshift-data:ExecuteStatement redshift-data:GetStatementResult redshift-data:ListDatabases redshift-data:ListSchemas redshift-data:ListSessions redshift-data:ListStatements redshift-data:ListTables

Service prefix	Actions
refactor-spaces	refactor-spaces:CreateApplication refactor-spaces:CreateEnvironment refactor-spaces:CreateRoute refactor-spaces:CreateService refactor-spaces:DeleteApplication refactor-spaces:DeleteEnvironment refactor-spaces:DeleteResourcePolicy refactor-spaces:DeleteRoute refactor-spaces:DeleteService refactor-spaces:GetApplication refactor-spaces:GetEnvironment refactor-spaces:GetResourcePolicy refactor-spaces:GetRoute refactor-spaces:GetService refactor-spaces:ListApplications refactor-spaces:ListEnvironmentVpcs refactor-spaces:ListEnvironments refactor-spaces:ListRoutes refactor-spaces:ListServices refactor-spaces:PutResourcePolicy refactor-spaces:UpdateRoute

Service prefix	Actions
rekognition	rekognition:AssociateFaces
	rekognition:CompareFaces
	rekognition:CopyProjectVersion
	rekognition:CreateCollection
	rekognition:CreateDataset
	rekognition:CreateFaceLivenessSession
	rekognition:CreateProject
	rekognition:CreateProjectVersion
	rekognition:CreateStreamProcessor
	rekognition:CreateUser
	rekognition:DeleteCollection
	rekognition:DeleteDataset
	rekognition:DeleteFaces
	rekognition:DeleteProject
	rekognition:DeleteProjectPolicy
	rekognition:DeleteProjectVersion
	rekognition:DeleteStreamProcessor
	rekognition:DeleteUser
	rekognition:DescribeCollection
	rekognition:DescribeDataset
	rekognition:DescribeProjectVersions

Service prefix	Actions
	rekognition:DescribeProjects
	rekognition:DescribeStreamProcessor
	rekognition:DetectCustomLabels
	rekognition:DetectFaces
	rekognition:DetectLabels
	rekognition:DetectModerationLabels
	rekognition:DetectProtectiveEquipment
	rekognition:DetectText
	rekognition:DisassociateFaces
	rekognition:DistributeDatasetEntries
	rekognition:GetCelebrityInfo
	rekognition:GetCelebrityRecognition
	rekognition:GetContentModeration
	rekognition:GetFaceDetection
	rekognition:GetFaceLivenessSessionResults
	rekognition:GetFaceSearch
	rekognition:GetLabelDetection
	rekognition:GetMediaAnalysisJob
	rekognition:GetPersonTracking
	rekognition:GetSegmentDetection
	rekognition:GetTextDetection

Service prefix	Actions
	rekognition:IndexFaces
	rekognition:ListCollections
	rekognition:ListDatasetEntries
	rekognition:ListDatasetLabels
	rekognition:ListFaces
	rekognition:ListMediaAnalysisJobs
	rekognition:ListProjectPolicies
	rekognition:ListStreamProcessors
	rekognition:ListUsers
	rekognition:PutProjectPolicy
	rekognition:RecognizeCelebrities
	rekognition:SearchFaces
	rekognition:SearchFacesByImage
	rekognition:SearchUsers
	rekognition:SearchUsersByImage
	rekognition:StartCelebrityRecognition
	rekognition:StartContentModeration
	rekognition:StartFaceDetection
	rekognition:StartFaceLivenessSession
	rekognition:StartFaceSearch
	rekognition:StartLabelDetection

Service prefix	Actions
	rekognition:StartMediaAnalysisJob rekognition:StartPersonTracking rekognition:StartProjectVersion rekognition:StartSegmentDetection rekognition:StartStreamProcessor rekognition:StartTextDetection rekognition:StopProjectVersion rekognition:StopStreamProcessor rekognition:UpdateDatasetEntries rekognition:UpdateStreamProcessor

Service prefix	Actions
resiliencehub	resiliencehub:AcceptResourceGroupingRecommendations resiliencehub:AddDraftAppVersionResourceMappings resiliencehub:BatchUpdateRecommendationStatus resiliencehub:CreateApp resiliencehub:CreateAppVersionAppComponent resiliencehub:CreateAppVersionResource resiliencehub:CreateRecommendationTemplate resiliencehub:CreateResiliencyPolicy resiliencehub:DeleteApp resiliencehub:DeleteAppAssessment resiliencehub:DeleteAppInputSource resiliencehub:DeleteAppVersionAppComponent resiliencehub:DeleteAppVersionResource resiliencehub:DeleteRecommendationTemplate resiliencehub:DeleteResiliencyPolicy resiliencehub:DescribeApp resiliencehub:DescribeAppAssessment resiliencehub:DescribeAppVersion resiliencehub:DescribeAppVersionAppComponent resiliencehub:DescribeAppVersionResource resiliencehub:DescribeAppVersionResourcesResolutionStatus

Service prefix	Actions
	resiliencehub:DescribeAppVersionTemplate resiliencehub:DescribeDraftAppVersionResourcesImportStatus resiliencehub:DescribeMetricsExport resiliencehub:DescribeResiliencyPolicy resiliencehub:DescribeResourceGroupingRecommendationTask resiliencehub:ImportResourcesToDraftAppVersion resiliencehub:ListAlarmRecommendations resiliencehub:ListAppAssessmentComplianceDrifts resiliencehub:ListAppAssessmentResourceDrifts resiliencehub:ListAppAssessments resiliencehub:ListAppComponentCompliances resiliencehub:ListAppComponentRecommendations resiliencehub:ListAppInputSources resiliencehub:ListAppVersionAppComponents resiliencehub:ListAppVersionResourceMappings resiliencehub:ListAppVersionResources resiliencehub:ListAppVersions resiliencehub:ListApps resiliencehub:ListMetrics resiliencehub:ListRecommendationTemplates resiliencehub:ListResiliencyPolicies

Service prefix	Actions
	resiliencehub:ListResourceGroupingRecommendations resiliencehub:ListSopRecommendations resiliencehub:ListSuggestedResiliencyPolicies resiliencehub:ListTestRecommendations resiliencehub:ListUnsupportedAppVersionResources resiliencehub:PublishAppVersion resiliencehub:PutDraftAppVersionTemplate resiliencehub:RejectResourceGroupingRecommendations resiliencehub:RemoveDraftAppVersionResourceMappings resiliencehub:ResolveAppVersionResources resiliencehub:StartAppAssessment resiliencehub:StartResourceGroupingRecommendationTask resiliencehub:UpdateApp resiliencehub:UpdateAppVersion resiliencehub:UpdateAppVersionAppComponent resiliencehub:UpdateAppVersionResource resiliencehub:UpdateResiliencyPolicy

Service prefix	Actions
resource-explorer-2	resource-explorer-2:AssociateDefaultView
	resource-explorer-2:BatchGetView
	resource-explorer-2:CreateIndex
	resource-explorer-2:CreateResourceExplorerSetup
	resource-explorer-2:CreateView
	resource-explorer-2:DeleteIndex
	resource-explorer-2:DeleteResourceExplorerSetup
	resource-explorer-2:DeleteView
	resource-explorer-2:DisassociateDefaultView
	resource-explorer-2:GetAccountLevelServiceConfiguration
	resource-explorer-2:GetDefaultView
	resource-explorer-2:GetIndex
	resource-explorer-2:GetManagedView
	resource-explorer-2:GetResourceExplorerSetup
	resource-explorer-2:GetServiceIndex
	resource-explorer-2:GetServiceView
	resource-explorer-2:ListIndexes
	resource-explorer-2:ListIndexesForMembers
	resource-explorer-2:ListManagedViews
	resource-explorer-2:ListServiceIndexes
	resource-explorer-2:ListServiceViews

Service prefix	Actions
	resource-explorer-2:ListStreamingAccessForServices resource-explorer-2:ListSupportedResourceTypes resource-explorer-2:ListViews resource-explorer-2:Search resource-explorer-2:UpdateIndexType resource-explorer-2:UpdateView

Service prefix	Actions
resource-groups	resource-groups:CancelTagSyncTask resource-groups:GetAccountSettings resource-groups:GetGroup resource-groups:GetGroupConfiguration resource-groups:GetGroupQuery resource-groups:GetTagSyncTask resource-groups:GroupResources resource-groups:ListGroupResources resource-groups:ListGroupingStatuses resource-groups:ListGroups resource-groups:ListTagSyncTasks resource-groups:PutGroupConfiguration resource-groups:SearchResources resource-groups:StartTagSyncTask resource-groups:UngroupResources resource-groups:UpdateAccountSettings resource-groups:UpdateGroup resource-groups:UpdateGroupQuery

Service prefix	Actions
robomaker	robomaker:BatchDeleteWorlds
	robomaker:BatchDescribeSimulationJob
	robomaker:CancelDeploymentJob
	robomaker:CancelSimulationJob
	robomaker:CancelSimulationJobBatch
	robomaker:CancelWorldExportJob
	robomaker:CancelWorldGenerationJob
	robomaker:CreateDeploymentJob
	robomaker:CreateFleet
	robomaker:CreateRobot
	robomaker:CreateRobotApplication
	robomaker:CreateRobotApplicationVersion
	robomaker:CreateSimulationApplication
	robomaker:CreateSimulationApplicationVersion
	robomaker:CreateSimulationJob
	robomaker:CreateWorldExportJob
	robomaker:CreateWorldGenerationJob
	robomaker:CreateWorldTemplate
	robomaker:DeleteFleet
	robomaker:DeleteRobot
	robomaker:DeleteRobotApplication

Service prefix	Actions
	robomaker:DeleteSimulationApplication
	robomaker:DeleteWorldTemplate
	robomaker:DeregisterRobot
	robomaker:DescribeDeploymentJob
	robomaker:DescribeFleet
	robomaker:DescribeRobot
	robomaker:DescribeRobotApplication
	robomaker:DescribeSimulationApplication
	robomaker:DescribeSimulationJob
	robomaker:DescribeSimulationJobBatch
	robomaker:DescribeWorld
	robomaker:DescribeWorldExportJob
	robomaker:DescribeWorldGenerationJob
	robomaker:DescribeWorldTemplate
	robomaker:GetWorldTemplateBody
	robomaker:ListDeploymentJobs
	robomaker:ListFleets
	robomaker:ListRobotApplications
	robomaker:ListRobots
	robomaker:ListSimulationApplications
	robomaker:ListSimulationJobBatches

Service prefix	Actions
	robomaker:ListSimulationJobs
	robomaker:ListWorldExportJobs
	robomaker:ListWorldGenerationJobs
	robomaker:ListWorldTemplates
	robomaker:ListWorlds
	robomaker:RegisterRobot
	robomaker:RestartSimulationJob
	robomaker:StartSimulationJobBatch
	robomaker:SyncDeploymentJob
	robomaker:UpdateRobotApplication
	robomaker:UpdateSimulationApplication
	robomaker:UpdateWorldTemplate

Service prefix	Actions
rolesanywhere	rolesanywhere:CreateProfile rolesanywhere:CreateTrustAnchor rolesanywhere:DeleteAttributeMapping rolesanywhere:DeleteCrl rolesanywhere:DeleteProfile rolesanywhere:DeleteTrustAnchor rolesanywhere:DisableCrl rolesanywhere:DisableProfile rolesanywhere:DisableTrustAnchor rolesanywhere:EnableCrl rolesanywhere:EnableProfile rolesanywhere:EnableTrustAnchor rolesanywhere:GetCrl rolesanywhere:GetProfile rolesanywhere:GetSubject rolesanywhere:GetTrustAnchor rolesanywhere:ImportCrl rolesanywhere:ListCrls rolesanywhere:ListProfiles rolesanywhere:ListSubjects rolesanywhere:ListTrustAnchors

Service prefix	Actions
	rolesanywhere:PutAttributeMapping rolesanywhere:PutNotificationSettings rolesanywhere:ResetNotificationSettings rolesanywhere:UpdateCrl rolesanywhere:UpdateProfile rolesanywhere:UpdateTrustAnchor

Service prefix	Actions
route53	route53:ActivateKeySigningKey
	route53:AssociateVPCWithHostedZone
	route53:ChangeCidrCollection
	route53:ChangeResourceRecordSets
	route53:CreateCidrCollection
	route53:CreateHealthCheck
	route53:CreateHostedZone
	route53:CreateKeySigningKey
	route53:CreateQueryLoggingConfig
	route53:CreateReusableDelegationSet
	route53:CreateTrafficPolicy
	route53:CreateTrafficPolicyInstance
	route53:CreateTrafficPolicyVersion
	route53:CreateVPCAssociationAuthorization
	route53:DeactivateKeySigningKey
	route53:DeleteCidrCollection
	route53:DeleteHealthCheck
	route53:DeleteHostedZone
	route53:DeleteKeySigningKey
	route53:DeleteQueryLoggingConfig
	route53:DeleteReusableDelegationSet

Service prefix	Actions
	route53:DeleteTrafficPolicy
	route53:DeleteTrafficPolicyInstance
	route53:DeleteVPCAssociationAuthorization
	route53:DisableHostedZoneDNSSEC
	route53:DisassociateVPCFromHostedZone
	route53:EnableHostedZoneDNSSEC
	route53:GetAccountLimit
	route53:GetChange
	route53:GetCheckerIpRanges
	route53:GetDNSSEC
	route53:GetGeoLocation
	route53:GetHealthCheck
	route53:GetHealthCheckCount
	route53:GetHealthCheckLastFailureReason
	route53:GetHealthCheckStatus
	route53:GetHostedZone
	route53:GetHostedZoneCount
	route53:GetHostedZoneLimit
	route53:GetQueryLoggingConfig
	route53:GetReusableDelegationSet
	route53:GetReusableDelegationSetLimit

Service prefix	Actions
	route53:GetTrafficPolicy
	route53:GetTrafficPolicyInstance
	route53:GetTrafficPolicyInstanceCount
	route53:ListCidrBlocks
	route53:ListCidrCollections
	route53:ListCidrLocations
	route53:ListGeoLocations
	route53:ListHealthChecks
	route53:ListHostedZones
	route53:ListHostedZonesByName
	route53:ListHostedZonesByVPC
	route53:ListQueryLoggingConfigs
	route53:ListResourceRecordSets
	route53:ListReusableDelegationSets
	route53:ListTrafficPolicies
	route53:ListTrafficPolicyInstances
	route53:ListTrafficPolicyInstancesByHostedZone
	route53:ListTrafficPolicyInstancesByPolicy
	route53:ListTrafficPolicyVersions
	route53:ListVPCAssociationAuthorizations
	route53:TestDNSAnswer

Service prefix	Actions
	route53:UpdateHealthCheck route53:UpdateHostedZoneComment route53:UpdateTrafficPolicyComment route53:UpdateTrafficPolicyInstance

Service prefix	Actions
route53-recovery-control-config	route53-recovery-control-config:CreateCluster route53-recovery-control-config:CreateControlPanel route53-recovery-control-config:CreateRoutingControl route53-recovery-control-config:CreateSafetyRule route53-recovery-control-config:DeleteCluster route53-recovery-control-config:DeleteControlPanel route53-recovery-control-config:DeleteRoutingControl route53-recovery-control-config:DeleteSafetyRule route53-recovery-control-config:DescribeCluster route53-recovery-control-config:DescribeControlPanel route53-recovery-control-config:DescribeRoutingControl route53-recovery-control-config:DescribeSafetyRule route53-recovery-control-config:GetResourcePolicy route53-recovery-control-config:ListAssociatedRoute53HealthChecks route53-recovery-control-config:ListClusters route53-recovery-control-config:ListControlPanels route53-recovery-control-config:ListRoutingControls route53-recovery-control-config:ListSafetyRules route53-recovery-control-config:UpdateCluster route53-recovery-control-config:UpdateControlPanel

Service prefix	Actions
	route53-recovery-control-config:UpdateRoutingControl route53-recovery-control-config:UpdateSafetyRule

Service prefix	Actions
route53-recovery-readiness	route53-recovery-readiness:CreateCell route53-recovery-readiness:CreateCrossAccountAuthorization route53-recovery-readiness:CreateReadinessCheck route53-recovery-readiness:CreateRecoveryGroup route53-recovery-readiness:CreateResourceSet route53-recovery-readiness:DeleteCell route53-recovery-readiness:DeleteCrossAccountAuthorization route53-recovery-readiness:DeleteReadinessCheck route53-recovery-readiness:DeleteRecoveryGroup route53-recovery-readiness:DeleteResourceSet route53-recovery-readiness:GetArchitectureRecommendations route53-recovery-readiness:GetCell route53-recovery-readiness:GetCellReadinessSummary route53-recovery-readiness:GetReadinessCheck route53-recovery-readiness:GetReadinessCheckResourceStatus route53-recovery-readiness:GetReadinessCheckStatus route53-recovery-readiness:GetRecoveryGroup route53-recovery-readiness:GetRecoveryGroupReadinessSummary route53-recovery-readiness:GetResourceSet route53-recovery-readiness:ListCells route53-recovery-readiness:ListCrossAccountAuthorizations

Service prefix	Actions
	route53-recovery-readiness:ListReadinessChecks
	route53-recovery-readiness:ListRecoveryGroups
	route53-recovery-readiness:ListResourceSets
	route53-recovery-readiness:ListRules
	route53-recovery-readiness:UpdateCell
	route53-recovery-readiness:UpdateReadinessCheck
	route53-recovery-readiness:UpdateRecoveryGroup
	route53-recovery-readiness:UpdateResourceSet

Service prefix	Actions
route53resolver	route53resolver:AssociateFirewallRuleGroup route53resolver:AssociateResolverEndpointIpAddress route53resolver:AssociateResolverQueryLogConfig route53resolver:AssociateResolverRule route53resolver:CreateFirewallDomainList route53resolver:CreateFirewallRule route53resolver:CreateFirewallRuleGroup route53resolver:CreateResolverEndpoint route53resolver:CreateResolverQueryLogConfig route53resolver:CreateResolverRule route53resolver:DeleteFirewallDomainList route53resolver:DeleteFirewallRule route53resolver:DeleteFirewallRuleGroup route53resolver:DeleteOutpostResolver route53resolver:DeleteResolverEndpoint route53resolver:DeleteResolverQueryLogConfig route53resolver:DeleteResolverRule route53resolver:DisassociateFirewallRuleGroup route53resolver:DisassociateResolverEndpointIpAddress route53resolver:DisassociateResolverQueryLogConfig route53resolver:DisassociateResolverRule

Service prefix	Actions
	route53resolver:GetFirewallConfig route53resolver:GetFirewallDomainList route53resolver:GetFirewallRuleGroup route53resolver:GetFirewallRuleGroupAssociation route53resolver:GetFirewallRuleGroupPolicy route53resolver:GetOutpostResolver route53resolver:GetResolverConfig route53resolver:GetResolverDnssecConfig route53resolver:GetResolverEndpoint route53resolver:GetResolverQueryLogConfig route53resolver:GetResolverQueryLogConfigAssociation route53resolver:GetResolverQueryLogConfigPolicy route53resolver:GetResolverRule route53resolver:GetResolverRuleAssociation route53resolver:GetResolverRulePolicy route53resolver:ImportFirewallDomains route53resolver:ListFirewallConfigs route53resolver:ListFirewallDomainLists route53resolver:ListFirewallDomains route53resolver:ListFirewallRuleGroupAssociations route53resolver:ListFirewallRuleGroups

Service prefix	Actions
	route53resolver:ListFirewallRuleTypes route53resolver:ListFirewallRules route53resolver:ListOutpostResolvers route53resolver:ListResolverConfigs route53resolver:ListResolverDnssecConfigs route53resolver:ListResolverEndpointIpAddresses route53resolver:ListResolverEndpoints route53resolver:ListResolverQueryLogConfigAssociations route53resolver:ListResolverQueryLogConfigs route53resolver:ListResolverRuleAssociations route53resolver:ListResolverRules route53resolver:PutFirewallRuleGroupPolicy route53resolver:PutResolverQueryLogConfigPolicy route53resolver:UpdateFirewallConfig route53resolver:UpdateFirewallDomains route53resolver:UpdateFirewallRule route53resolver:UpdateFirewallRuleGroupAssociation route53resolver:UpdateOutpostResolver route53resolver:UpdateResolverConfig route53resolver:UpdateResolverDnssecConfig route53resolver:UpdateResolverEndpoint

Service prefix	Actions
	route53resolver:UpdateResolverRule
rum	rum:BatchCreateRumMetricDefinitions rum:BatchDeleteRumMetricDefinitions rum:BatchGetRumMetricDefinitions rum:CreateAppMonitor rum:DeleteAppMonitor rum:DeleteResourcePolicy rum:DeleteRumMetricsDestination rum:GetAppMonitor rum:GetAppMonitorData rum:GetResourcePolicy rum:ListAppMonitors rum:ListRumMetricsDestinations rum:PutResourcePolicy rum:PutRumMetricsDestination rum:UpdateAppMonitor rum:UpdateRumMetricDefinition

Service prefix	Actions
s3	s3:AssociateAccessGrantsIdentityCenter s3:CreateAccessGrant s3:CreateAccessGrantsInstance s3:CreateAccessGrantsLocation s3:CreateAccessPoint s3:CreateAccessPointForObjectLambda s3:CreateBucket s3:CreateBucketMetadataTableConfiguration s3:CreateJob s3:CreateMultiRegionAccessPoint s3:DeleteAccessGrant s3:DeleteAccessGrantsInstance s3:DeleteAccessGrantsInstanceResourcePolicy s3:DeleteAccessGrantsLocation s3:DeleteAccessPoint s3:DeleteAccessPointForObjectLambda s3:DeleteAccessPointPolicy s3:DeleteAccessPointPolicyForObjectLambda s3:DeleteBucket s3:DeleteBucketMetadataTableConfiguration s3:DeleteBucketPolicy

Service prefix	Actions
	s3:DeleteBucketWebsite s3:DeleteMultiRegionAccessPoint s3:DeleteStorageLensConfiguration s3:DescribeJob s3:DescribeMultiRegionAccessPointOperation s3:DissociateAccessGrantsIdentityCenter s3:GetAccelerateConfiguration s3:GetAccessGrant s3:GetAccessGrantsInstance s3:GetAccessGrantsInstanceForPrefix s3:GetAccessGrantsInstanceResourcePolicy s3:GetAccessGrantsLocation s3:GetAccessPoint s3:GetAccessPointConfigurationForObjectLambda s3:GetAccessPointForObjectLambda s3:GetAccessPointPolicy s3:GetAccessPointPolicyForObjectLambda s3:GetAccessPointPolicyStatus s3:GetAccessPointPolicyStatusForObjectLambda s3:GetAccountPublicAccessBlock s3:GetAnalyticsConfiguration

Service prefix	Actions
	s3:GetBucketAbac s3:GetBucketAcl s3:GetBucketCORS s3:GetBucketLocation s3:GetBucketLogging s3:GetBucketNotification s3:GetBucketObjectLockConfiguration s3:GetBucketOwnershipControls s3:GetBucketPolicy s3:GetBucketPolicyStatus s3:GetBucketPublicAccessBlock s3:GetBucketRequestPayment s3:GetBucketVersioning s3:GetBucketWebsite s3:GetDataAccess s3:GetEncryptionConfiguration s3:GetIntelligentTieringConfiguration s3:GetInventoryConfiguration s3:GetLifecycleConfiguration s3:GetMetricsConfiguration s3:GetMultiRegionAccessPoint

Service prefix	Actions
	s3:GetMultiRegionAccessPointPolicy s3:GetMultiRegionAccessPointPolicyStatus s3:GetMultiRegionAccessPointRoutes s3:GetReplicationConfiguration s3:GetStorageLensConfiguration s3:GetStorageLensDashboard s3:ListAccessGrants s3:ListAccessGrantsInstances s3:ListAccessGrantsLocations s3:ListAccessPoints s3:ListAccessPointsForObjectLambda s3:ListAllMyBuckets s3:ListBucketMultipartUploads s3:ListCallerAccessGrants s3:ListJobs s3:ListMultiRegionAccessPoints s3:ListStorageLensConfigurations s3:PutAccelerateConfiguration s3:PutAccessGrantsInstanceResourcePolicy s3:PutAccessPointConfigurationForObjectLambda s3:PutAccessPointPolicy

Service prefix	Actions
	s3:PutAccessPointPolicyForObjectLambda s3:PutAccountPublicAccessBlock s3:PutAnalyticsConfiguration s3:PutBucketAbac s3:PutBucketAcl s3:PutBucketCORS s3:PutBucketLogging s3:PutBucketNotification s3:PutBucketObjectLockConfiguration s3:PutBucketOwnershipControls s3:PutBucketPolicy s3:PutBucketPublicAccessBlock s3:PutBucketRequestPayment s3:PutBucketVersioning s3:PutBucketWebsite s3:PutEncryptionConfiguration s3:PutIntelligentTieringConfiguration s3:PutInventoryConfiguration s3:PutLifecycleConfiguration s3:PutMetricsConfiguration s3:PutMultiRegionAccessPointPolicy

Service prefix	Actions
	s3:PutReplicationConfiguration s3:PutStorageLensConfiguration s3:SubmitMultiRegionAccessPointRoutes s3:UpdateAccessGrantsLocation s3:UpdateBucketMetadataAnnotationTableConfiguration s3:UpdateBucketMetadataJournalTableConfiguration s3:UpdateJobPriority s3:UpdateJobStatus
s3-outposts	s3-outposts:CreateEndpoint s3-outposts:DeleteEndpoint s3-outposts:ListEndpoints s3-outposts:ListOutpostsWithS3 s3-outposts:ListSharedEndpoints

Service prefix	Actions
sagemaker-geospatial	sagemaker-geospatial:DeleteEarthObservationJob sagemaker-geospatial:DeleteVectorEnrichmentJob sagemaker-geospatial:ExportEarthObservationJob sagemaker-geospatial:ExportVectorEnrichmentJob sagemaker-geospatial:GetEarthObservationJob sagemaker-geospatial:GetRasterDataCollection sagemaker-geospatial:GetTile sagemaker-geospatial:GetVectorEnrichmentJob sagemaker-geospatial:ListEarthObservationJobs sagemaker-geospatial:ListRasterDataCollections sagemaker-geospatial:ListVectorEnrichmentJobs sagemaker-geospatial:SearchRasterDataCollection sagemaker-geospatial:StartEarthObservationJob sagemaker-geospatial:StartVectorEnrichmentJob sagemaker-geospatial:StopEarthObservationJob sagemaker-geospatial:StopVectorEnrichmentJob

Service prefix	Actions
savingsplans	savingsplans:CreateSavingsPlan savingsplans:DeleteQueuedSavingsPlan savingsplans:DescribeSavingsPlanRates savingsplans:DescribeSavingsPlans savingsplans:DescribeSavingsPlansOfferingRates savingsplans:DescribeSavingsPlansOfferings savingsplans:ReturnSavingsPlan

Service prefix	Actions
schemas	schemas:CreateDiscoverer schemas:CreateRegistry schemas:CreateSchema schemas:DeleteDiscoverer schemas:DeleteRegistry schemas:DeleteResourcePolicy schemas:DeleteSchema schemas:DeleteSchemaVersion schemas:DescribeCodeBinding schemas:DescribeDiscoverer schemas:DescribeRegistry schemas:DescribeSchema schemas:ExportSchema schemas:GetCodeBindingSource schemas:GetDiscoveredSchema schemas:GetResourcePolicy schemas:ListDiscoverers schemas:ListRegistries schemas:ListSchemaVersions schemas:ListSchemas schemas:PutCodeBinding

Service prefix	Actions
	schemas:PutResourcePolicy schemas:SearchSchemas schemas:StartDiscoverer schemas:StopDiscoverer schemas:UpdateDiscoverer schemas:UpdateRegistry schemas:UpdateSchema
sdb	sdb:CreateDomain sdb:DeleteDomain sdb:DomainMetadata sdb:ListDomains

Service prefix	Actions
secretsmanager	secretsmanager:CancelRotateSecret secretsmanager:CreateSecret secretsmanager:DeleteResourcePolicy secretsmanager:DeleteSecret secretsmanager:DescribeSecret secretsmanager:GetRandomPassword secretsmanager:GetResourcePolicy secretsmanager:GetSecretValue secretsmanager:ListSecretVersionIds secretsmanager:ListSecrets secretsmanager:PutResourcePolicy secretsmanager:PutSecretValue secretsmanager:RemoveRegionsFromReplication secretsmanager:ReplicateSecretToRegions secretsmanager:RestoreSecret secretsmanager:RotateSecret secretsmanager:StopReplicationToReplica secretsmanager:UpdateSecret secretsmanager:ValidateResourcePolicy

Service prefix	Actions
securityhub	securityhub:AcceptAdministratorInvitation
	securityhub:AcceptInvitation
	securityhub:BatchDeleteAutomationRules
	securityhub:BatchDisableStandards
	securityhub:BatchEnableStandards
	securityhub:BatchGetAutomationRules
	securityhub:BatchGetConfigurationPolicyAssociations
	securityhub:BatchGetEnabledRegionsV2
	securityhub:BatchGetSecurityControls
	securityhub:BatchGetStandardsControlAssociations
	securityhub:BatchImportFindings
	securityhub:BatchUpdateAutomationRules
	securityhub:BatchUpdateFindings
	securityhub:BatchUpdateStandardsControlAssociations
	securityhub:ConnectorRegistrationsV2
	securityhub:CreateActionTarget
	securityhub:CreateAggregatorV2
	securityhub:CreateAutomationRule
	securityhub:CreateAutomationRuleV2
	securityhub:CreateConfigurationPolicy
	securityhub:CreateConnector

Service prefix	Actions
	securityhub:CreateConnectorV2
	securityhub:CreateFindingAggregator
	securityhub:CreateInsight
	securityhub:CreateMembers
	securityhub:CreateTicketV2
	securityhub:DeclineInvitations
	securityhub:DeleteActionTarget
	securityhub:DeleteAggregatorV2
	securityhub:DeleteAutomationRuleV2
	securityhub:DeleteConfigurationPolicy
	securityhub:DeleteConnector
	securityhub:DeleteConnectorV2
	securityhub:DeleteFindingAggregator
	securityhub:DeleteInsight
	securityhub:DeleteInvitations
	securityhub:DeleteMembers
	securityhub:DescribeActionTargets
	securityhub:DescribeHub
	securityhub:DescribeOrganizationConfiguration
	securityhub:DescribeProducts
	securityhub:DescribeSecurityHubV2

Service prefix	Actions
	securityhub:DescribeStandards
	securityhub:DisableImportFindingsForProduct
	securityhub:DisableOrganizationAdminAccount
	securityhub:DisableSecurityHub
	securityhub:DisableSecurityHubFeatureV2
	securityhub:DisableSecurityHubV2
	securityhub:DisassociateFromAdministratorAccount
	securityhub:DisassociateFromMasterAccount
	securityhub:DisassociateMembers
	securityhub:EnableImportFindingsForProduct
	securityhub:EnableOrganizationAdminAccount
	securityhub:EnableSecurityHub
	securityhub:EnableSecurityHubFeatureV2
	securityhub:GetAdministratorAccount
	securityhub:GetAggregatorV2
	securityhub:GetAutomationRuleV2
	securityhub:GetConfigurationPolicy
	securityhub:GetConfigurationPolicyAssociation
	securityhub:GetConnector
	securityhub:GetConnectorV2
	securityhub:GetCoverageStatisticsV2

Service prefix	Actions
	securityhub:GetEnabledStandards
	securityhub:GetFindingAggregator
	securityhub:GetFindingHistory
	securityhub:GetFindings
	securityhub:GetInsightResults
	securityhub:GetInsights
	securityhub:GetInvitationsCount
	securityhub:GetMasterAccount
	securityhub:GetMembers
	securityhub:GetSecurityControlDefinition
	securityhub:InviteMembers
	securityhub:ListAggregatorsV2
	securityhub:ListAutomationRules
	securityhub:ListAutomationRulesV2
	securityhub:ListConfigurationPolicies
	securityhub:ListConfigurationPolicyAssociations
	securityhub:ListConnectors
	securityhub:ListConnectorsV2
	securityhub:ListEnabledProductsForImport
	securityhub:ListFindingAggregators
	securityhub:ListInvitations

Service prefix	Actions
	securityhub:ListMembers
	securityhub:ListOrganizationAdminAccounts
	securityhub:ListSecurityControlDefinitions
	securityhub:ListStandardsControlAssociations
	securityhub:StartConfigurationPolicyAssociation
	securityhub:StartConfigurationPolicyDisassociation
	securityhub:UpdateActionTarget
	securityhub:UpdateAggregatorV2
	securityhub:UpdateAutomationRuleV2
	securityhub:UpdateConfigurationPolicy
	securityhub:UpdateConnector
	securityhub:UpdateConnectorV2
	securityhub:UpdateFindingAggregator
	securityhub:UpdateFindings
	securityhub:UpdateInsight
	securityhub:UpdateOrganizationConfiguration
	securityhub:UpdateSecurityControl
	securityhub:UpdateSecurityHubConfiguration

Service prefix	Actions
securitylake	securitylake:CreateAwsLogSource securitylake:CreateCustomLogSource securitylake:CreateDataLakeExceptionSubscription securitylake:CreateDataLakeOrganizationConfiguration securitylake:CreateSubscriber securitylake:CreateSubscriberNotification securitylake:DeleteAwsLogSource securitylake:DeleteCustomLogSource securitylake:DeleteDataLakeExceptionSubscription securitylake:DeleteDataLakeOrganizationConfiguration securitylake:DeleteSubscriber securitylake:DeleteSubscriberNotification securitylake:DeregisterDataLakeDelegatedAdministrator securitylake:GetDataLakeExceptionSubscription securitylake:GetDataLakeOrganizationConfiguration securitylake:GetDataLakeSources securitylake:GetSubscriber securitylake:ListDataLakes securitylake:ListLogSources securitylake:ListSubscribers securitylake:RegisterDataLakeDelegatedAdministrator

Service prefix	Actions
	securitylake:UpdateDataLakeExceptionSubscription securitylake:UpdateSubscriber securitylake:UpdateSubscriberNotification
serverlessrepo	serverlessrepo:CreateApplication serverlessrepo:CreateApplicationVersion serverlessrepo:CreateCloudFormationChangeSet serverlessrepo:CreateCloudFormationTemplate serverlessrepo:DeleteApplication serverlessrepo:GetApplication serverlessrepo:GetApplicationPolicy serverlessrepo:GetCloudFormationTemplate serverlessrepo:ListApplicationDependencies serverlessrepo:ListApplicationVersions serverlessrepo:ListApplications serverlessrepo:PutApplicationPolicy serverlessrepo:UnshareApplication serverlessrepo:UpdateApplication

Service prefix	Actions
servicecatalog	servicecatalog:AcceptPortfolioShare servicecatalog:AssociateBudgetWithResource servicecatalog:AssociatePrincipalWithPortfolio servicecatalog:AssociateProductWithPortfolio servicecatalog:AssociateServiceActionWithProvisioningArtifact servicecatalog:BatchAssociateServiceActionWithProvisioningArtifact servicecatalog:BatchDisassociateServiceActionFromProvisioningArtifact servicecatalog:CopyProduct servicecatalog:CreateAttributeGroup servicecatalog:CreateConstraint servicecatalog:CreatePortfolio servicecatalog:CreatePortfolioShare servicecatalog:CreateProduct servicecatalog:CreateProvisionedProductPlan servicecatalog:CreateProvisioningArtifact servicecatalog:CreateServiceAction servicecatalog:DeleteAttributeGroup servicecatalog:DeleteConstraint servicecatalog:DeletePortfolio servicecatalog:DeletePortfolioShare

Service prefix	Actions
	servicecatalog:DeleteProduct servicecatalog:DeleteProvisionedProductPlan servicecatalog:DeleteProvisioningArtifact servicecatalog:DeleteServiceAction servicecatalog:DescribeConstraint servicecatalog:DescribeCopyProductStatus servicecatalog:DescribePortfolio servicecatalog:DescribePortfolioShareStatus servicecatalog:DescribePortfolioShares servicecatalog:DescribeProduct servicecatalog:DescribeProductAsAdmin servicecatalog:DescribeProductView servicecatalog:DescribeProvisionedProduct servicecatalog:DescribeProvisionedProductPlan servicecatalog:DescribeProvisioningArtifact servicecatalog:DescribeProvisioningParameters servicecatalog:DescribeRecord servicecatalog:DescribeServiceAction servicecatalog:DescribeServiceActionExecutionParameters servicecatalog:DisableAWSOrganizationsAccess servicecatalog:DisassociateBudgetFromResource

Service prefix	Actions
	servicecatalog:DisassociatePrincipalFromPortfolio servicecatalog:DisassociateProductFromPortfolio servicecatalog:DisassociateServiceActionFromProvisioningArtifact servicecatalog:EnableAWSOrganizationsAccess servicecatalog:ExecuteProvisionedProductPlan servicecatalog:ExecuteProvisionedProductServiceAction servicecatalog:GetAWSOrganizationsAccessStatus servicecatalog:GetProvisionedProductOutputs servicecatalog:ImportAsProvisionedProduct servicecatalog:ListAcceptedPortfolioShares servicecatalog:ListAttributeGroups servicecatalog:ListBudgetsForResource servicecatalog:ListConstraintsForPortfolio servicecatalog:ListLaunchPaths servicecatalog:ListOrganizationPortfolioAccess servicecatalog:ListPortfolioAccess servicecatalog:ListPortfolios servicecatalog:ListPortfoliosForProduct servicecatalog:ListPrincipalsForPortfolio servicecatalog:ListProvisionedProductPlans servicecatalog:ListProvisioningArtifacts

Service prefix	Actions
	servicecatalog:ListProvisioningArtifactsForServiceAction servicecatalog:ListRecordHistory servicecatalog:ListServiceActions servicecatalog:ListServiceActionsForProvisioningArtifact servicecatalog:ListStackInstancesForProvisionedProduct servicecatalog:NotifyProvisionProductEngineWorkflowResult servicecatalog:NotifyTerminateProvisionedProductEngine WorkflowResult servicecatalog:NotifyUpdateProvisionedProductEngineWor kflowResult servicecatalog:ProvisionProduct servicecatalog:RejectPortfolioShare servicecatalog:ScanProvisionedProducts servicecatalog:SearchProducts servicecatalog:SearchProductsAsAdmin servicecatalog:SearchProvisionedProducts servicecatalog:TerminateProvisionedProduct servicecatalog:UpdateConstraint servicecatalog:UpdatePortfolio servicecatalog:UpdatePortfolioShare servicecatalog:UpdateProduct servicecatalog:UpdateProvisionedProduct

Service prefix	Actions
	servicecatalog:UpdateProvisionedProductProperties servicecatalog:UpdateProvisioningArtifact servicecatalog:UpdateServiceAction

Service prefix	Actions
servicediscovery	servicediscovery:CreateHttpNamespace
	servicediscovery:CreatePrivateDnsNamespace
	servicediscovery:CreatePublicDnsNamespace
	servicediscovery:CreateService
	servicediscovery:DeleteNamespace
	servicediscovery:DeleteService
	servicediscovery:DeleteServiceAttributes
	servicediscovery:DeregisterInstance
	servicediscovery:GetInstance
	servicediscovery:GetInstancesHealthStatus
	servicediscovery:GetNamespace
	servicediscovery:GetOperation
	servicediscovery:GetService
	servicediscovery:ListInstances
	servicediscovery:ListNamespaces
	servicediscovery:ListOperations
	servicediscovery:ListServices
	servicediscovery:RegisterInstance
	servicediscovery:UpdateHttpNamespace
	servicediscovery:UpdateInstanceCustomHealthStatus
	servicediscovery:UpdatePrivateDnsNamespace

Service prefix	Actions
	servicediscovery:UpdatePublicDnsNamespace servicediscovery:UpdateService servicediscovery:UpdateServiceAttributes

Service prefix	Actions
servicequotas	servicequotas:AssociateServiceQuotaTemplate servicequotas:CreateSupportCase servicequotas:DeleteServiceQuotaIncreaseRequestFromTemplate servicequotas:DisassociateServiceQuotaTemplate servicequotas:GetAWSDefaultServiceQuota servicequotas:GetAssociationForServiceQuotaTemplate servicequotas:GetAutoManagementConfiguration servicequotas:GetQuotaUtilizationReport servicequotas:GetRequestedServiceQuotaChange servicequotas:GetServiceQuota servicequotas:GetServiceQuotaIncreaseRequestFromTemplate servicequotas:ListAWSDefaultServiceQuotas servicequotas:ListRequestedServiceQuotaChangeHistory servicequotas:ListRequestedServiceQuotaChangeHistoryByQuota servicequotas:ListServiceQuotaIncreaseRequestsInTemplate servicequotas:ListServiceQuotas servicequotas:ListServices servicequotas:PutServiceQuotaIncreaseRequestIntoTemplate servicequotas:RequestServiceQuotaIncrease servicequotas:StartAutoManagement servicequotas:StartQuotaUtilizationReport

Service prefix	Actions
	servicequotas:StopAutoManagement servicequotas:UpdateAutoManagement

Service prefix	Actions
ses	ses:BatchGetMetricData
	ses:CloneReceiptRuleSet
	ses:CreateAddonInstance
	ses:CreateAddonSubscription
	ses:CreateAddressList
	ses:CreateAddressListImportJob
	ses:CreateArchive
	ses:CreateConfigurationSet
	ses:CreateConfigurationSetEventDestination
	ses:CreateConfigurationSetTrackingOptions
	ses:CreateContact
	ses:CreateContactList
	ses:CreateCustomVerificationEmailTemplate
	ses:CreateDedicatedIpPool
	ses:CreateDeliverabilityTestReport
	ses:CreateEmailIdentity
	ses:CreateEmailIdentityPolicy
	ses:CreateEmailTemplate
	ses:CreateImportJob
	ses:CreateIngressPoint
	ses:CreateMultiRegionEndpoint

Service prefix	Actions
	ses:CreateReceiptFilter
	ses:CreateReceiptRule
	ses:CreateReceiptRuleSet
	ses:CreateRelay
	ses:CreateRuleSet
	ses:CreateTemplate
	ses:CreateTenant
	ses:CreateTenantResourceAssociation
	ses:CreateTrafficPolicy
	ses:DeleteAddonInstance
	ses:DeleteAddonSubscription
	ses:DeleteAddressList
	ses:DeleteArchive
	ses:DeleteConfigurationSet
	ses:DeleteConfigurationSetEventDestination
	ses:DeleteConfigurationSetTrackingOptions
	ses:DeleteContact
	ses:DeleteContactList
	ses:DeleteCustomVerificationEmailTemplate
	ses:DeleteDedicatedIpPool
	ses:DeleteEmailIdentity

Service prefix	Actions
	ses:DeleteEmailIdentityPolicy
	ses:DeleteEmailTemplate
	ses:DeleteIdentity
	ses:DeleteIdentityPolicy
	ses:DeleteIngressPoint
	ses:DeleteMultiRegionEndpoint
	ses:DeleteReceiptFilter
	ses:DeleteReceiptRule
	ses:DeleteReceiptRuleSet
	ses:DeleteRelay
	ses:DeleteRuleSet
	ses:DeleteSuppressedDestination
	ses:DeleteTemplate
	ses:DeleteTenant
	ses:DeleteTenantResourceAssociation
	ses:DeleteTrafficPolicy
	ses:DeleteVerifiedEmailAddress
	ses:DeregisterMemberFromAddressList
	ses:DescribeActiveReceiptRuleSet
	ses:DescribeConfigurationSet
	ses:DescribeReceiptRule

Service prefix	Actions
	ses:DescribeReceiptRuleSet
	ses:GetAccount
	ses:GetAccountSendingEnabled
	ses:GetAddonInstance
	ses:GetAddonSubscription
	ses:GetAddressList
	ses:GetArchive
	ses:GetArchiveExport
	ses:GetArchiveMessage
	ses:GetArchiveMessageContent
	ses:GetArchiveSearch
	ses:GetArchiveSearchResults
	ses:GetBlacklistReports
	ses:GetConfigurationSet
	ses:GetConfigurationSetEventDestinations
	ses:GetContact
	ses:GetContactList
	ses:GetCustomVerificationEmailTemplate
	ses:GetDedicatedIp
	ses:GetDedicatedIpPool
	ses:GetDedicatedIps

Service prefix	Actions
	ses:GetDeliverabilityDashboardOptions
	ses:GetDeliverabilityTestReport
	ses:GetDomainDeliverabilityCampaign
	ses:GetDomainStatisticsReport
	ses:GetEmailAddressInsights
	ses:GetEmailIdentity
	ses:GetEmailIdentityPolicies
	ses:GetEmailTemplate
	ses:GetIdentityDkimAttributes
	ses:GetIdentityMailFromDomainAttributes
	ses:GetIdentityNotificationAttributes
	ses:GetIdentityPolicies
	ses:GetIdentityVerificationAttributes
	ses:GetImportJob
	ses:GetIngressPoint
	ses:GetMemberOfAddressList
	ses:GetMessageInsights
	ses:GetMultiRegionEndpoint
	ses:GetRelay
	ses:GetRuleSet
	ses:GetSendQuota

Service prefix	Actions
	ses:GetSendStatistics
	ses:GetSuppressedDestination
	ses:GetTemplate
	ses:GetTenant
	ses:GetTrafficPolicy
	ses:ListAddonInstances
	ses:ListAddonSubscriptions
	ses:ListAddressListImportJobs
	ses:ListAddressLists
	ses:ListArchiveExports
	ses:ListArchiveSearches
	ses:ListArchives
	ses:ListConfigurationSets
	ses:ListContactLists
	ses:ListContacts
	ses:ListCustomVerificationEmailTemplates
	ses:ListDedicatedIpPools
	ses:ListDeliverabilityTestReports
	ses:ListDomainDeliverabilityCampaigns
	ses:ListEmailIdentities
	ses:ListEmailTemplates

Service prefix	Actions
	ses:ListExportJobs
	ses:ListIdentities
	ses:ListIdentityPolicies
	ses:ListImportJobs
	ses:ListIngressPoints
	ses:ListMembersOfAddressList
	ses:ListMultiRegionEndpoints
	ses:ListReceiptFilters
	ses:ListReceiptRuleSets
	ses:ListRecommendations
	ses:ListRelays
	ses:ListReputationEntities
	ses:ListResourceTenants
	ses:ListRuleSets
	ses:ListSuppressedDestinations
	ses:ListTemplates
	ses:ListTenantResources
	ses:ListTenants
	ses:ListTrafficPolicies
	ses:ListVerifiedEmailAddresses
	ses:PutAccountDedicatedIpWarmupAttributes

Service prefix	Actions
	ses:PutAccountDetails
	ses:PutAccountSendingAttributes
	ses:PutAccountSuppressionAttributes
	ses:PutAccountVdmAttributes
	ses:PutConfigurationSetArchivingOptions
	ses:PutConfigurationSetDeliveryOptions
	ses:PutConfigurationSetReputationOptions
	ses:PutConfigurationSetSendingOptions
	ses:PutConfigurationSetSuppressionOptions
	ses:PutConfigurationSetTrackingOptions
	ses:PutConfigurationSetVdmOptions
	ses:PutDedicatedIpInPool
	ses:PutDedicatedIpPoolScalingAttributes
	ses:PutDedicatedIpWarmupAttributes
	ses:PutDeliverabilityDashboardOption
	ses:PutEmailIdentityConfigurationSetAttributes
	ses:PutEmailIdentityDkimAttributes
	ses:PutEmailIdentityDkimSigningAttributes
	ses:PutEmailIdentityFeedbackAttributes
	ses:PutEmailIdentityMailFromAttributes
	ses:PutIdentityPolicy

Service prefix	Actions
	ses:PutSuppressedDestination
	ses:PutTenantSuppressionAttributes
	ses:RegisterMemberToAddressList
	ses:ReorderReceiptRuleSet
	ses:SendBounce
	ses:SendCustomVerificationEmail
	ses:SetActiveReceiptRuleSet
	ses:SetIdentityDkimEnabled
	ses:SetIdentityFeedbackForwardingEnabled
	ses:SetIdentityHeadersInNotificationsEnabled
	ses:SetIdentityMailFromDomain
	ses:SetIdentityNotificationTopic
	ses:SetReceiptRulePosition
	ses:StartArchiveExport
	ses:StartArchiveSearch
	ses:StopArchiveExport
	ses:StopArchiveSearch
	ses:TestRenderEmailTemplate
	ses:TestRenderTemplate
	ses:UpdateAccountSendingEnabled
	ses:UpdateArchive

Service prefix	Actions
	ses:UpdateConfigurationSetEventDestination
	ses:UpdateConfigurationSetReputationMetricsEnabled
	ses:UpdateConfigurationSetSendingEnabled
	ses:UpdateConfigurationSetTrackingOptions
	ses:UpdateContact
	ses:UpdateContactList
	ses:UpdateCustomVerificationEmailTemplate
	ses:UpdateEmailIdentityPolicy
	ses:UpdateEmailTemplate
	ses:UpdateIngressPoint
	ses:UpdateReceiptRule
	ses:UpdateRelay
	ses:UpdateRuleSet
	ses:UpdateTemplate
	ses:UpdateTrafficPolicy
	ses:VerifyDomainDkim
	ses:VerifyDomainIdentity
	ses:VerifyEmailAddress
	ses:VerifyEmailIdentity

Service prefix	Actions
shield	shield:AssociateDRTLogBucket shield:AssociateHealthCheck shield:AssociateProactiveEngagementDetails shield:CreateProtection shield:CreateProtectionGroup shield:CreateSubscription shield:DeleteProtection shield:DeleteProtectionGroup shield:DeleteSubscription shield:DescribeAttack shield:DescribeAttackStatistics shield:DescribeDRTAccess shield:DescribeEmergencyContactSettings shield:DescribeProtection shield:DescribeProtectionGroup shield:DescribeSubscription shield:DisableApplicationLayerAutomaticResponse shield:DisableProactiveEngagement shield:DisassociateDRTLogBucket shield:DisassociateDRTRole shield:DisassociateHealthCheck

Service prefix	Actions
	shield:EnableApplicationLayerAutomaticResponse shield:EnableProactiveEngagement shield:GetSubscriptionState shield:ListAttacks shield:ListProtectionGroups shield:ListProtections shield:ListResourcesInProtectionGroup shield:UpdateApplicationLayerAutomaticResponse shield:UpdateEmergencyContactSettings shield:UpdateProtectionGroup shield:UpdateSubscription

Service prefix	Actions
signer	signer:AddProfilePermission
	signer:CancelSigningProfile
	signer:DescribeSigningJob
	signer:GetSigningPlatform
	signer:GetSigningProfile
	signer:ListProfilePermissions
	signer:ListSigningJobs
	signer:ListSigningPlatforms
	signer:ListSigningProfiles
	signer:PutSigningProfile
	signer:RemoveProfilePermission
	signer:RevokeSignature
	signer:RevokeSigningProfile
	signer:SignPayload
	signer:StartSigningJob

Service prefix	Actions
simspaceweaver	simspaceweaver:CreateSnapshot simspaceweaver:DeleteApp simspaceweaver:DeleteSimulation simspaceweaver:DescribeApp simspaceweaver:DescribeSimulation simspaceweaver:ListApps simspaceweaver:ListSimulations simspaceweaver:StartApp simspaceweaver:StartClock simspaceweaver:StartSimulation simspaceweaver:StopApp simspaceweaver:StopClock simspaceweaver:StopSimulation

Service prefix	Actions
sms	sms:CreateApp
	sms:CreateReplicationJob
	sms:DeleteApp
	sms:DeleteAppLaunchConfiguration
	sms:DeleteAppReplicationConfiguration
	sms:DeleteAppValidationConfiguration
	sms:DeleteReplicationJob
	sms:DeleteServerCatalog
	sms:DisassociateConnector
	sms:GenerateChangeSet
	sms:GenerateTemplate
	sms:GetApp
	sms:GetAppLaunchConfiguration
	sms:GetAppReplicationConfiguration
	sms:GetAppValidationConfiguration
	sms:GetAppValidationOutput
	sms:GetConnectors
	sms:GetReplicationJobs
	sms:GetReplicationRuns
	sms:GetServers
	sms:ImportAppCatalog

Service prefix	Actions
	sms:ImportServerCatalog
	sms:LaunchApp
	sms:ListApps
	sms:NotifyAppValidationOutput
	sms:PutAppLaunchConfiguration
	sms:PutAppReplicationConfiguration
	sms:PutAppValidationConfiguration
	sms:StartAppReplication
	sms:StartOnDemandAppReplication
	sms:StartOnDemandReplicationRun
	sms:StopAppReplication
	sms:TerminateApp
	sms:UpdateApp
	sms:UpdateReplicationJob

Service prefix	Actions
sms-voice	sms-voice:AssociateProtectConfiguration sms-voice:CreateConfigurationSet sms-voice:CreateConfigurationSetEventDestination sms-voice:CreateEventDestination sms-voice:CreateOptOutList sms-voice:CreatePool sms-voice:CreateProtectConfiguration sms-voice:CreateRegistration sms-voice:CreateRegistrationAssociation sms-voice:CreateRegistrationAttachment sms-voice:CreateRegistrationVersion sms-voice:CreateVerifiedDestinationNumber sms-voice:DeleteAccountDefaultProtectConfiguration sms-voice:DeleteConfigurationSet sms-voice:DeleteConfigurationSetEventDestination sms-voice:DeleteDefaultMessageType sms-voice:DeleteDefaultSenderId sms-voice:DeleteEventDestination sms-voice:DeleteKeyword sms-voice:DeleteMediaMessageSpendLimitOverride sms-voice:DeleteOptOutList

Service prefix	Actions
	sms-voice:DeleteOptedOutNumber sms-voice:DeletePool sms-voice:DeleteProtectConfiguration sms-voice:DeleteProtectConfigurationRuleSetNumberOverride sms-voice:DeleteRegistration sms-voice:DeleteRegistrationAttachment sms-voice:DeleteResourcePolicy sms-voice:DeleteTextMessageSpendLimitOverride sms-voice:DeleteVerifiedDestinationNumber sms-voice:DeleteVoiceMessageSpendLimitOverride sms-voice:DescribeAccountAttributes sms-voice:DescribeAccountLimits sms-voice:DescribeConfigurationSets sms-voice:DescribeKeywords sms-voice:DescribeOptOutLists sms-voice:DescribeOptedOutNumbers sms-voice:DescribePhoneNumbers sms-voice:DescribePools sms-voice:DescribeProtectConfigurations sms-voice:DescribeRegistrationAttachments sms-voice:DescribeRegistrationFieldDefinitions

Service prefix	Actions
	sms-voice:DescribeRegistrationFieldValues sms-voice:DescribeRegistrationSectionDefinitions sms-voice:DescribeRegistrationTypeDefinitions sms-voice:DescribeRegistrationVersions sms-voice:DescribeRegistrations sms-voice:DescribeSenderIds sms-voice:DescribeSpendLimits sms-voice:DescribeVerifiedDestinationNumbers sms-voice:DisassociateOriginationIdentity sms-voice:DisassociateProtectConfiguration sms-voice:DiscardRegistrationVersion sms-voice:GetConfigurationSetEventDestinations sms-voice:GetProtectConfigurationCountryRuleSet sms-voice:GetResourcePolicy sms-voice:ListConfigurationSets sms-voice:ListPoolOriginationIdentities sms-voice:ListProtectConfigurationRuleSetNumberOverrides sms-voice:ListRegistrationAssociations sms-voice:PutKeyword sms-voice:PutOptedOutNumber sms-voice:PutProtectConfigurationRuleSetNumberOverride

Service prefix	Actions
	sms-voice:PutResourcePolicy sms-voice:ReleasePhoneNumber sms-voice:ReleaseSenderId sms-voice:RequestPhoneNumber sms-voice:RequestSenderId sms-voice:SendDestinationNumberVerificationCode sms-voice:SetAccountDefaultProtectConfiguration sms-voice:SetDefaultMessageFeedbackEnabled sms-voice:SetDefaultMessageType sms-voice:SetDefaultSenderId sms-voice:SetMediaMessageSpendLimitOverride sms-voice:SetTextMessageSpendLimitOverride sms-voice:SetVoiceMessageSpendLimitOverride sms-voice:SubmitRegistrationVersion sms-voice:UpdateConfigurationSetEventDestination sms-voice:UpdateEventDestination sms-voice:UpdatePhoneNumber sms-voice:UpdatePool sms-voice:UpdateProtectConfiguration sms-voice:UpdateProtectConfigurationCountryRuleSet sms-voice:UpdateSenderId

Service prefix	Actions
snowball	snowball:CancelCluster
	snowball:CancelJob
	snowball:CreateAddress
	snowball:CreateCluster
	snowball:CreateJob
	snowball:CreateLongTermPricing
	snowball:CreateReturnShippingLabel
	snowball:DescribeAddress
	snowball:DescribeAddresses
	snowball:DescribeCluster
	snowball:DescribeJob
	snowball:DescribeReturnShippingLabel
	snowball:GetJobManifest
	snowball:GetJobUnlockCode
	snowball:GetSnowballUsage
	snowball:GetSoftwareUpdates
	snowball:ListClusterJobs
	snowball:ListClusters
	snowball:ListCompatibleImages
	snowball:ListJobs
	snowball:ListLongTermPricing

Service prefix	Actions
	snowball:ListPickupLocations snowball:ListServiceVersions snowball:UpdateCluster snowball:UpdateJob snowball:UpdateJobShipmentState snowball:UpdateLongTermPricing
sqs	sqs:AddPermission sqs:CancelMessageMoveTask sqs:CreateQueue sqs:DeleteQueue sqs:PurgeQueue sqs:RemovePermission sqs:SetQueueAttributes

Service prefix	Actions
ssm	ssm:AssociateOpsItemRelatedItem
	ssm:CancelCommand
	ssm:CancelMaintenanceWindowExecution
	ssm:CreateActivation
	ssm:CreateAssociation
	ssm:CreateAssociationBatch
	ssm:CreateDocument
	ssm:CreateMaintenanceWindow
	ssm:CreateOpsItem
	ssm:CreateOpsMetadata
	ssm:CreatePatchBaseline
	ssm:CreateResourceDataSync
	ssm:DeleteActivation
	ssm:DeleteAssociation
	ssm:DeleteCloudConnector
	ssm:DeleteDocument
	ssm:DeleteInventory
	ssm:DeleteMaintenanceWindow
	ssm:DeleteOpsItem
	ssm:DeleteOpsMetadata
	ssm:DeleteParameter

Service prefix	Actions
	ssm:DeleteParameters
	ssm:DeletePatchBaseline
	ssm:DeleteResourceDataSync
	ssm:DeleteResourcePolicy
	ssm:DeregisterManagedInstance
	ssm:DeregisterPatchBaselineForPatchGroup
	ssm:DeregisterTargetFromMaintenanceWindow
	ssm:DeregisterTaskFromMaintenanceWindow
	ssm:DescribeActivations
	ssm:DescribeAssociation
	ssm:DescribeAssociationExecutionTargets
	ssm:DescribeAssociationExecutions
	ssm:DescribeAutomationExecutions
	ssm:DescribeAutomationStepExecutions
	ssm:DescribeAvailablePatches
	ssm:DescribeDocument
	ssm:DescribeDocumentParameters
	ssm:DescribeDocumentPermission
	ssm:DescribeEffectiveInstanceAssociations
	ssm:DescribeEffectivePatchesForPatchBaseline
	ssm:DescribeInstanceAssociationsStatus

Service prefix	Actions
	ssm:DescribeInstanceInformation
	ssm:DescribeInstancePatchStates
	ssm:DescribeInstancePatchStatesForPatchGroup
	ssm:DescribeInstancePatches
	ssm:DescribeInstanceProperties
	ssm:DescribeInventoryDeletions
	ssm:DescribeMaintenanceWindowExecutionTaskInvocations
	ssm:DescribeMaintenanceWindowExecutionTasks
	ssm:DescribeMaintenanceWindowExecutions
	ssm:DescribeMaintenanceWindowSchedule
	ssm:DescribeMaintenanceWindowTargets
	ssm:DescribeMaintenanceWindowTasks
	ssm:DescribeMaintenanceWindows
	ssm:DescribeMaintenanceWindowsForTarget
	ssm:DescribeOpsItems
	ssm:DescribeParameters
	ssm:DescribePatchBaselines
	ssm:DescribePatchGroupState
	ssm:DescribePatchGroups
	ssm:DescribePatchProperties
	ssm:DescribeSessions

Service prefix	Actions
	ssm:DisassociateOpsItemRelatedItem
	ssm:GetAccessToken
	ssm:GetAutomationExecution
	ssm:GetCalendarState
	ssm:GetCloudConnector
	ssm:GetCommandInvocation
	ssm:GetConnectionStatus
	ssm:GetDefaultPatchBaseline
	ssm:GetDeployablePatchSnapshotForInstance
	ssm:GetDocument
	ssm:GetExecutionPreview
	ssm:GetInventory
	ssm:GetInventorySchema
	ssm:GetMaintenanceWindow
	ssm:GetMaintenanceWindowExecution
	ssm:GetMaintenanceWindowExecutionTask
	ssm:GetMaintenanceWindowExecutionTaskInvocation
	ssm:GetMaintenanceWindowTask
	ssm:GetOpsItem
	ssm:GetOpsMetadata
	ssm:GetOpsSummary

Service prefix	Actions
	ssm:GetParameter
	ssm:GetParameterHistory
	ssm:GetParameters
	ssm:GetParametersByPath
	ssm:GetPatchBaseline
	ssm:GetPatchBaselineForPatchGroup
	ssm:GetResourcePolicies
	ssm:GetServiceSetting
	ssm:LabelParameterVersion
	ssm:ListAssociationVersions
	ssm:ListAssociations
	ssm:ListCloudConnectors
	ssm:ListCommandInvocations
	ssm:ListCommands
	ssm:ListComplianceItems
	ssm:ListComplianceSummaries
	ssm:ListDocumentMetadataHistory
	ssm:ListDocumentVersions
	ssm:ListDocuments
	ssm:ListInstanceAssociations
	ssm:ListInventoryEntries

Service prefix	Actions
	ssm:ListNodes
	ssm:ListNodesSummary
	ssm:ListOpsItemEvents
	ssm:ListOpsItemRelatedItems
	ssm:ListOpsMetadata
	ssm:ListResourceComplianceSummaries
	ssm:ListResourceDataSync
	ssm:ModifyDocumentPermission
	ssm:PutComplianceItems
	ssm:PutInventory
	ssm:PutParameter
	ssm:PutResourcePolicy
	ssm:RegisterDefaultPatchBaseline
	ssm:RegisterManagedInstance
	ssm:RegisterPatchBaselineForPatchGroup
	ssm:RegisterTargetWithMaintenanceWindow
	ssm:RegisterTaskWithMaintenanceWindow
	ssm:ResetServiceSetting
	ssm:ResumeSession
	ssm:SendAutomationSignal
	ssm:SendCommand

Service prefix	Actions
	ssm:StartAssociationsOnce
	ssm:StartAutomationExecution
	ssm:StartChangeRequestExecution
	ssm:StartSession
	ssm:StopAutomationExecution
	ssm:TerminateSession
	ssm:UnlabelParameterVersion
	ssm:UpdateAssociation
	ssm:UpdateAssociationStatus
	ssm:UpdateCloudConnector
	ssm:UpdateDocument
	ssm:UpdateDocumentDefaultVersion
	ssm:UpdateDocumentMetadata
	ssm:UpdateInstanceInformation
	ssm:UpdateMaintenanceWindow
	ssm:UpdateMaintenanceWindowTarget
	ssm:UpdateMaintenanceWindowTask
	ssm:UpdateManagedInstanceRole
	ssm:UpdateOpsItem
	ssm:UpdateOpsMetadata
	ssm:UpdatePatchBaseline

Service prefix	Actions
	ssm:UpdateResourceDataSync ssm:UpdateServiceSetting ssm:ValidateCloudConnector

Service prefix	Actions
ssm-incidents	ssm-incidents:BatchGetIncidentFindings ssm-incidents:CreateReplicationSet ssm-incidents:CreateResponsePlan ssm-incidents:CreateTimelineEvent ssm-incidents:DeleteIncidentRecord ssm-incidents:DeleteReplicationSet ssm-incidents:DeleteResourcePolicy ssm-incidents:DeleteResponsePlan ssm-incidents:DeleteTimelineEvent ssm-incidents:GetIncidentRecord ssm-incidents:GetReplicationSet ssm-incidents:GetResourcePolicies ssm-incidents:GetResponsePlan ssm-incidents:GetTimelineEvent ssm-incidents:ListIncidentFindings ssm-incidents:ListIncidentRecords ssm-incidents:ListRelatedItems ssm-incidents:ListReplicationSets ssm-incidents:ListResponsePlans ssm-incidents:ListTimelineEvents ssm-incidents:PutResourcePolicy

Service prefix	Actions
	ssm-incidents:StartIncident ssm-incidents:UpdateDeletionProtection ssm-incidents:UpdateIncidentRecord ssm-incidents:UpdateRelatedItems ssm-incidents:UpdateReplicationSet ssm-incidents:UpdateResponsePlan ssm-incidents:UpdateTimelineEvent

Service prefix	Actions
ssm-sap	ssm-sap:BackupDatabase
	ssm-sap:DeleteResourcePermission
	ssm-sap:DeregisterApplication
	ssm-sap:GetApplication
	ssm-sap:GetComponent
	ssm-sap:GetConfigurationCheckOperation
	ssm-sap:GetDatabase
	ssm-sap:GetOperation
	ssm-sap:GetResourcePermission
	ssm-sap:ListApplications
	ssm-sap:ListComponents
	ssm-sap:ListConfigurationCheckDefinitions
	ssm-sap:ListConfigurationCheckOperations
	ssm-sap:ListDatabases
	ssm-sap:ListOperationEvents
	ssm-sap:ListOperations
	ssm-sap:ListSubCheckResults
	ssm-sap:ListSubCheckRuleResults
	ssm-sap:PutResourcePermission
	ssm-sap:RegisterApplication
	ssm-sap:RestoreDatabase

Service prefix	Actions
	ssm-sap:StartApplication ssm-sap:StartApplicationRefresh ssm-sap:StartConfigurationChecks ssm-sap:StopApplication ssm-sap:UpdateApplicationSettings ssm-sap:UpdateHANABackupSettings

Service prefix	Actions
states	states:CreateActivity states:CreateStateMachine states:CreateStateMachineAlias states:DeleteActivity states:DeleteStateMachine states:DeleteStateMachineAlias states:DeleteStateMachineVersion states:DescribeActivity states:DescribeExecution states:DescribeMapRun states:DescribeStateMachine states:DescribeStateMachineAlias states:DescribeStateMachineForExecution states:GetExecutionHistory states:ListActivities states:ListExecutions states:ListMapRuns states:ListStateMachineAliases states:ListStateMachineVersions states:ListStateMachines states:SendTaskFailure

Service prefix	Actions
	states:SendTaskHeartbeat states:SendTaskSuccess states:StartExecution states:StopExecution states:UpdateMapRun states:UpdateStateMachine states:UpdateStateMachineAlias states:ValidateStateMachineDefinition
sts	sts:AssumeRole sts:AssumeRoleWithSAML sts:AssumeRoleWithWebIdentity sts:DecodeAuthorizationMessage sts:GetAccessKeyInfo sts:GetCallerIdentity sts:GetFederationToken sts:GetSessionToken sts:GetWebIdentityToken

Service prefix	Actions
swf	swf:DeleteActivityType swf:DeleteWorkflowType swf:DeprecateActivityType swf:DeprecateDomain swf:DeprecateWorkflowType swf:DescribeActivityType swf:DescribeDomain swf:DescribeWorkflowType swf:ListActivityTypes swf:ListDomains swf:ListWorkflowTypes swf:RegisterActivityType swf:RegisterDomain swf:RegisterWorkflowType swf:UndeprecateActivityType swf:UndeprecateDomain swf:UndeprecateWorkflowType

Service prefix	Actions
synthetics	synthetics:AssociateResource
	synthetics:CreateCanary
	synthetics:CreateGroup
	synthetics:DeleteCanary
	synthetics:DeleteGroup
	synthetics:DescribeCanaries
	synthetics:DescribeCanariesLastRun
	synthetics:DescribeRuntimeVersions
	synthetics:DisassociateResource
	synthetics:GetCanary
	synthetics:GetCanaryRuns
	synthetics:GetGroup
	synthetics:ListAssociatedGroups
	synthetics:ListGroupResources
	synthetics:ListGroups
	synthetics:StartCanary
	synthetics:StartCanaryDryRun
	synthetics:StopCanary
	synthetics:UpdateCanary

Service prefix	Actions
tag	tag:DescribeReportCreation tag:GetComplianceSummary tag:GetResources tag:StartReportCreation

Service prefix	Actions
textract	textract:AnalyzeDocument
	textract:AnalyzeExpense
	textract:AnalyzeID
	textract:CreateAdapter
	textract:CreateAdapterVersion
	textract:DeleteAdapter
	textract:DeleteAdapterVersion
	textract:DetectDocumentText
	textract:GetAdapter
	textract:GetAdapterVersion
	textract:GetDocumentAnalysis
	textract:GetDocumentTextDetection
	textract:GetExpenseAnalysis
	textract:GetLendingAnalysis
	textract:GetLendingAnalysisSummary
	textract:ListAdapterVersions
	textract:ListAdapters
	textract:StartDocumentAnalysis
	textract:StartDocumentTextDetection
	textract:StartExpenseAnalysis
	textract:StartLendingAnalysis

Service prefix	Actions
	textract:UpdateAdapter

Service prefix	Actions
timestream	timestream:CancelQuery timestream:CreateDatabase timestream:CreateScheduledQuery timestream:CreateTable timestream:DeleteDatabase timestream:DeleteScheduledQuery timestream:DeleteTable timestream:DescribeAccountSettings timestream:DescribeDatabase timestream:DescribeScheduledQuery timestream:DescribeTable timestream:ExecuteScheduledQuery timestream:ListBatchLoadTasks timestream:ListDatabases timestream:ListScheduledQueries timestream:ListTables timestream:PrepareQuery timestream:UpdateAccountSettings timestream:UpdateDatabase timestream:UpdateScheduledQuery timestream:UpdateTable

Service prefix	Actions
tnb	tnb:CancelSolNetworkOperation
	tnb:CreateSolFunctionPackage
	tnb:CreateSolNetworkInstance
	tnb:CreateSolNetworkPackage
	tnb:DeleteSolFunctionPackage
	tnb:DeleteSolNetworkInstance
	tnb:DeleteSolNetworkPackage
	tnb:GetSolFunctionInstance
	tnb:GetSolFunctionPackage
	tnb:GetSolFunctionPackageContent
	tnb:GetSolFunctionPackageDescriptor
	tnb:GetSolNetworkInstance
	tnb:GetSolNetworkOperation
	tnb:GetSolNetworkPackage
	tnb:GetSolNetworkPackageContent
	tnb:GetSolNetworkPackageDescriptor
	tnb:InstantiateSolNetworkInstance
	tnb:ListSolFunctionInstances
	tnb:ListSolFunctionPackages
	tnb:ListSolNetworkInstances
	tnb:ListSolNetworkOperations

Service prefix	Actions
	tnb:ListSolNetworkPackages tnb:PutSolFunctionPackageContent tnb:PutSolNetworkPackageContent tnb:TerminateSolNetworkInstance tnb:UpdateSolFunctionPackage tnb:UpdateSolNetworkInstance tnb:UpdateSolNetworkPackage tnb:ValidateSolFunctionPackageContent tnb:ValidateSolNetworkPackageContent

Service prefix	Actions
transcribe	transcribe:CreateCallAnalyticsCategory transcribe:CreateLanguageModel transcribe:CreateMedicalVocabulary transcribe:CreateVocabulary transcribe:CreateVocabularyFilter transcribe:DeleteCallAnalyticsCategory transcribe:DeleteCallAnalyticsJob transcribe:DeleteLanguageModel transcribe:DeleteMedicalScribeJob transcribe:DeleteMedicalTranscriptionJob transcribe:DeleteMedicalVocabulary transcribe:DeleteTranscriptionJob transcribe:DeleteVocabulary transcribe:DeleteVocabularyFilter transcribe:DescribeLanguageModel transcribe:GetCallAnalyticsCategory transcribe:GetCallAnalyticsJob transcribe:GetMedicalScribeJob transcribe:GetMedicalTranscriptionJob transcribe:GetMedicalVocabulary transcribe:GetTranscriptionJob

Service prefix	Actions
	transcribe:GetVocabulary transcribe:GetVocabularyFilter transcribe:ListCallAnalyticsCategories transcribe:ListCallAnalyticsJobs transcribe:ListLanguageModels transcribe:ListMedicalScribeJobs transcribe:ListMedicalTranscriptionJobs transcribe:ListMedicalVocabularies transcribe:ListTranscriptionJobs transcribe:ListVocabularies transcribe:ListVocabularyFilters transcribe:StartCallAnalyticsJob transcribe:StartCallAnalyticsStreamTranscription transcribe:StartCallAnalyticsStreamTranscriptionWebSocket transcribe:StartMedicalScribeJob transcribe:StartMedicalStreamTranscription transcribe:StartMedicalStreamTranscriptionWebSocket transcribe:StartMedicalTranscriptionJob transcribe:StartStreamTranscription transcribe:StartStreamTranscriptionWebSocket transcribe:StartTranscriptionJob

Service prefix	Actions
	transcribe:UpdateCallAnalyticsCategory transcribe:UpdateMedicalVocabulary transcribe:UpdateVocabulary transcribe:UpdateVocabularyFilter

Service prefix	Actions
transfer	transfer:CreateAccess
	transfer:CreateAgreement
	transfer:CreateConnector
	transfer:CreateProfile
	transfer:CreateServer
	transfer:CreateUser
	transfer:CreateWebApp
	transfer:CreateWorkflow
	transfer:DeleteAccess
	transfer:DeleteAgreement
	transfer:DeleteCertificate
	transfer:DeleteConnector
	transfer:DeleteHostKey
	transfer:DeleteProfile
	transfer:DeleteServer
	transfer:DeleteSshPublicKey
	transfer:DeleteUser
	transfer:DeleteWebApp
	transfer:DeleteWebAppCustomization
	transfer:DeleteWorkflow
	transfer:DescribeAccess

Service prefix	Actions
	transfer:DescribeAgreement
	transfer:DescribeCertificate
	transfer:DescribeConnector
	transfer:DescribeExecution
	transfer:DescribeHostKey
	transfer:DescribeProfile
	transfer:DescribeSecurityPolicy
	transfer:DescribeServer
	transfer:DescribeUser
	transfer:DescribeWebApp
	transfer:DescribeWebAppCustomization
	transfer:DescribeWorkflow
	transfer:ImportCertificate
	transfer:ImportHostKey
	transfer:ImportSshPublicKey
	transfer:ListAccesses
	transfer:ListCertificates
	transfer:ListConnectors
	transfer:ListExecutions
	transfer:ListFileTransferResults
	transfer:ListHostKeys

Service prefix	Actions
	transfer:ListProfiles
	transfer:ListSecurityPolicies
	transfer:ListServers
	transfer:ListUsers
	transfer:ListWebApps
	transfer:ListWorkflows
	transfer:SendWorkflowStepState
	transfer:StartDirectoryListing
	transfer:StartFileTransfer
	transfer:StartRemoteDelete
	transfer:StartRemoteMove
	transfer:StartServer
	transfer:StopServer
	transfer:TestConnection
	transfer:TestIdentityProvider
	transfer:UpdateAccess
	transfer:UpdateAgreement
	transfer:UpdateCertificate
	transfer:UpdateConnector
	transfer:UpdateHostKey
	transfer:UpdateProfile

Service prefix	Actions
	transfer:UpdateServer transfer:UpdateUser transfer:UpdateWebApp transfer:UpdateWebAppCustomization
translate	translate:CreateParallelData translate:DeleteParallelData translate:DeleteTerminology translate:DescribeTextTranslationJob translate:GetParallelData translate:GetTerminology translate:ImportTerminology translate:ListLanguages translate:ListParallelData translate:ListTerminologies translate:ListTextTranslationJobs translate:StartTextTranslationJob translate:StopTextTranslationJob translate:TranslateDocument translate:TranslateText translate:UpdateParallelData

Service prefix	Actions
voiceid	voiceid:AssociateFraudster
	voiceid:CreateDomain
	voiceid:CreateWatchlist
	voiceid:DeleteDomain
	voiceid:DeleteFraudster
	voiceid:DeleteSpeaker
	voiceid:DeleteWatchlist
	voiceid:DescribeDomain
	voiceid:DescribeFraudster
	voiceid:DescribeFraudsterRegistrationJob
	voiceid:DescribeSpeaker
	voiceid:DescribeSpeakerEnrollmentJob
	voiceid:DescribeWatchlist
	voiceid:DisassociateFraudster
	voiceid:EvaluateSession
	voiceid:ListDomains
	voiceid:ListFraudsterRegistrationJobs
	voiceid:ListFraudsters
	voiceid:ListSpeakerEnrollmentJobs
	voiceid:ListSpeakers
	voiceid:ListWatchlists

Service prefix	Actions
	voiceid:OptOutSpeaker voiceid:StartFraudsterRegistrationJob voiceid:StartSpeakerEnrollmentJob voiceid:UpdateDomain voiceid:UpdateWatchlist

Service prefix	Actions
vpc-lattice	vpc-lattice:CreateAccessLogSubscription vpc-lattice:CreateListener vpc-lattice:CreateResourceConfiguration vpc-lattice:CreateResourceGateway vpc-lattice:CreateRule vpc-lattice:CreateService vpc-lattice:CreateServiceNetwork vpc-lattice:CreateServiceNetworkResourceAssociation vpc-lattice:CreateServiceNetworkServiceAssociation vpc-lattice:CreateServiceNetworkVpcAssociation vpc-lattice:CreateTargetGroup vpc-lattice:DeleteAccessLogSubscription vpc-lattice:DeleteAuthPolicy vpc-lattice:DeleteDomainVerification vpc-lattice:DeleteListener vpc-lattice:DeleteResourceConfiguration vpc-lattice:DeleteResourceEndpointAssociation vpc-lattice:DeleteResourceGateway vpc-lattice:DeleteResourcePolicy vpc-lattice:DeleteRule vpc-lattice:DeleteService

Service prefix	Actions
	vpc-lattice:DeleteServiceNetwork vpc-lattice:DeleteServiceNetworkResourceAssociation vpc-lattice:DeleteServiceNetworkServiceAssociation vpc-lattice:DeleteServiceNetworkVpcAssociation vpc-lattice:DeleteTargetGroup vpc-lattice:DeregisterTargets vpc-lattice:GetAccessLogSubscription vpc-lattice:GetAuthPolicy vpc-lattice:GetDomainVerification vpc-lattice:GetListener vpc-lattice:GetResourceConfiguration vpc-lattice:GetResourceGateway vpc-lattice:GetResourcePolicy vpc-lattice:GetRule vpc-lattice:GetService vpc-lattice:GetServiceNetwork vpc-lattice:GetServiceNetworkResourceAssociation vpc-lattice:GetServiceNetworkServiceAssociation vpc-lattice:GetServiceNetworkVpcAssociation vpc-lattice:GetTargetGroup vpc-lattice:ListAccessLogSubscriptions

Service prefix	Actions
	vpc-lattice:ListDomainVerifications vpc-lattice:ListListeners vpc-lattice:ListResourceConfigurations vpc-lattice:ListResourceEndpointAssociations vpc-lattice:ListResourceGateways vpc-lattice:ListRules vpc-lattice:ListServiceNetworkResourceAssociations vpc-lattice:ListServiceNetworkServiceAssociations vpc-lattice:ListServiceNetworkVpcAssociations vpc-lattice:ListServiceNetworkVpcEndpointAssociations vpc-lattice:ListServiceNetworks vpc-lattice:ListServices vpc-lattice:ListTargetGroups vpc-lattice:ListTargets vpc-lattice:PutAuthPolicy vpc-lattice:PutResourcePolicy vpc-lattice:RegisterTargets vpc-lattice:StartDomainVerification vpc-lattice:UpdateAccessLogSubscription vpc-lattice:UpdateListener vpc-lattice:UpdateResourceConfiguration

Service prefix	Actions
	vpc-lattice:UpdateResourceGateway vpc-lattice:UpdateRule vpc-lattice:UpdateService vpc-lattice:UpdateServiceNetwork vpc-lattice:UpdateServiceNetworkVpcAssociation vpc-lattice:UpdateTargetGroup

Service prefix	Actions
wafv2	wafv2:AssociateWebACL
	wafv2:CheckCapacity
	wafv2:CreateAPIKey
	wafv2:CreateIPSet
	wafv2:CreateRegexPatternSet
	wafv2:CreateRuleGroup
	wafv2:CreateWebACL
	wafv2:DeleteAPIKey
	wafv2:DeleteFirewallManagerRuleGroups
	wafv2:DeleteIPSet
	wafv2:DeleteLoggingConfiguration
	wafv2:DeletePermissionPolicy
	wafv2:DeleteRegexPatternSet
	wafv2:DeleteRuleGroup
	wafv2:DeleteWebACL
	wafv2:DescribeAllManagedProducts
	wafv2:DescribeManagedProductsByVendor
	wafv2:DescribeManagedRuleGroup
	wafv2:DisassociateWebACL
	wafv2:GenerateMobileSdkReleaseUrl
	wafv2:GetDecryptedAPIKey

Service prefix	Actions
	wafv2:GetIPSet
	wafv2:GetLoggingConfiguration
	wafv2:GetManagedRuleSet
	wafv2:GetMobileSdkRelease
	wafv2:GetRateBasedStatementManagedKeys
	wafv2:GetRegexPatternSet
	wafv2:GetRevenueStatistics
	wafv2:GetRevenueStatisticsSummary
	wafv2:GetRevenueStatisticsTimeSeries
	wafv2:GetRuleGroup
	wafv2:GetSampledRequests
	wafv2:GetWebACLForResource
	wafv2:ListAPIKeys
	wafv2:ListAvailableManagedRuleGroupVersions
	wafv2:ListAvailableManagedRuleGroups
	wafv2:ListIPSets
	wafv2:ListLoggingConfigurations
	wafv2:ListManagedRuleSets
	wafv2:ListMobileSdkReleases
	wafv2:ListRegexPatternSets
	wafv2:ListResourcesForWebACL

Service prefix	Actions
	wafv2:ListRuleGroups wafv2:ListSettlementRecords wafv2:ListWebACLs wafv2:PutLoggingConfiguration wafv2:PutManagedRuleSetVersions wafv2:UpdateIPSet wafv2:UpdateManagedRuleSetVersionExpiryDate wafv2:UpdateRegexPatternSet wafv2:UpdateRuleGroup wafv2:UpdateWebACL

Service prefix	Actions
wellarchitected	wellarchitected:AssociateLenses wellarchitected:AssociateProfiles wellarchitected:CreateLensShare wellarchitected:CreateLensVersion wellarchitected:CreateMilestone wellarchitected:CreateProfile wellarchitected:CreateProfileShare wellarchitected:CreateReviewTemplate wellarchitected:CreateWorkload wellarchitected:CreateWorkloadShare wellarchitected:DeleteLens wellarchitected:DeleteLensShare wellarchitected:DeleteProfile wellarchitected:DeleteProfileShare wellarchitected:DeleteReviewTemplate wellarchitected:DeleteTemplateShare wellarchitected:DeleteWorkload wellarchitected:DeleteWorkloadShare wellarchitected:DisassociateLenses wellarchitected:DisassociateProfiles wellarchitected:ExportLens

Service prefix	Actions
	wellarchitected:GetAnswer wellarchitected:GetConsolidatedReport wellarchitected:GetGlobalSettings wellarchitected:GetLens wellarchitected:GetLensReview wellarchitected:GetLensReviewReport wellarchitected:GetLensVersionDifference wellarchitected:GetMilestone wellarchitected:GetProfile wellarchitected:GetProfileTemplate wellarchitected:GetReviewTemplate wellarchitected:GetReviewTemplateAnswer wellarchitected:GetReviewTemplateLensReview wellarchitected:GetWorkload wellarchitected:ImportLens wellarchitected:ListAnswers wellarchitected:ListCheckDetails wellarchitected:ListCheckSummaries wellarchitected:ListLensReviewImprovements wellarchitected:ListLensReviews wellarchitected:ListLensShares

Service prefix	Actions
	wellarchitected:ListLenses wellarchitected:ListMilestones wellarchitected:ListNotifications wellarchitected:ListProfileNotifications wellarchitected:ListProfileShares wellarchitected:ListProfiles wellarchitected:ListReviewTemplateAnswers wellarchitected:ListReviewTemplates wellarchitected:ListShareInvitations wellarchitected:ListTemplateShares wellarchitected:ListWorkloadShares wellarchitected:ListWorkloads wellarchitected:UpdateAnswer wellarchitected:UpdateGlobalSettings wellarchitected:UpdateIntegration wellarchitected:UpdateLensReview wellarchitected:UpdateProfile wellarchitected:UpdateReviewTemplate wellarchitected:UpdateReviewTemplateLensReview wellarchitected:UpdateShareInvitation wellarchitected:UpdateWorkload

Service prefix	Actions
	wellarchitected:UpdateWorkloadShare wellarchitected:UpgradeLensReview wellarchitected:UpgradeProfileVersion wellarchitected:UpgradeReviewTemplateLensReview

Service prefix	Actions
wisdom	wisdom:CreateAssistant
	wisdom:CreateAssistantAssociation
	wisdom:CreateContent
	wisdom:CreateKnowledgeBase
	wisdom:CreateQuickResponse
	wisdom:CreateSession
	wisdom:DeleteAssistant
	wisdom:DeleteAssistantAssociation
	wisdom:DeleteContent
	wisdom:DeleteImportJob
	wisdom:DeleteKnowledgeBase
	wisdom:DeleteQuickResponse
	wisdom:GetAssistant
	wisdom:GetAssistantAssociation
	wisdom:GetContent
	wisdom:GetContentAssociation
	wisdom:GetContentSummary
	wisdom:GetImportJob
	wisdom:GetKnowledgeBase
	wisdom:GetRecommendations
	wisdom:GetSession

Service prefix	Actions
	wisdom:ListAssistantAssociations
	wisdom:ListAssistants
	wisdom:ListContentAssociations
	wisdom:ListContents
	wisdom:ListImportJobs
	wisdom:ListKnowledgeBases
	wisdom:ListQuickResponses
	wisdom:NotifyRecommendationsReceived
	wisdom:QueryAssistant
	wisdom:RemoveKnowledgeBaseTemplateUri
	wisdom:SearchContent
	wisdom:SearchQuickResponses
	wisdom:SearchSessions
	wisdom:StartContentUpload
	wisdom:StartImportJob
	wisdom:UpdateContent
	wisdom:UpdateKnowledgeBaseTemplateUri
	wisdom:UpdateQuickResponse
	wisdom:UpdateSession

Service prefix	Actions
worklink	worklink:AssociateDomain worklink:AssociateWebsiteAuthorizationProvider worklink:AssociateWebsiteCertificateAuthority worklink:CreateFleet worklink:DeleteFleet worklink:DescribeAuditStreamConfiguration worklink:DescribeCompanyNetworkConfiguration worklink:DescribeDevice worklink:DescribeDevicePolicyConfiguration worklink:DescribeDomain worklink:DescribeFleetMetadata worklink:DescribeIdentityProviderConfiguration worklink:DescribeWebsiteCertificateAuthority worklink:DisassociateDomain worklink:DisassociateWebsiteAuthorizationProvider worklink:DisassociateWebsiteCertificateAuthority worklink:ListDevices worklink:ListDomains worklink:ListFleets worklink:ListWebsiteAuthorizationProviders worklink:ListWebsiteCertificateAuthorities

Service prefix	Actions
	worklink:RestoreDomainAccess worklink:RevokeDomainAccess worklink:SignOutUser worklink:UpdateAuditStreamConfiguration worklink:UpdateCompanyNetworkConfiguration worklink:UpdateDevicePolicyConfiguration worklink:UpdateDomainMetadata worklink:UpdateFleetMetadata worklink:UpdateIdentityProviderConfiguration

Service prefix	Actions
workspaces	workspaces:AcceptAccountLinkInvitation
	workspaces:AssociateConnectionAlias
	workspaces:AssociateIpGroups
	workspaces:AssociateWorkspaceApplication
	workspaces:CopyWorkspaceImage
	workspaces:CreateAccountLinkInvitation
	workspaces:CreateConnectClientAddIn
	workspaces:CreateConnectionAlias
	workspaces:CreateIpGroup
	workspaces:CreateStandbyWorkspaces
	workspaces:CreateUpdatedWorkspaceImage
	workspaces:CreateWorkspaceBundle
	workspaces:CreateWorkspaceImage
	workspaces:CreateWorkspaces
	workspaces:CreateWorkspacesPool
	workspaces:DeleteAccountLinkInvitation
	workspaces:DeleteClientBranding
	workspaces:DeleteConnectClientAddIn
	workspaces:DeleteConnectionAlias
	workspaces:DeleteIpGroup
	workspaces:DeleteWorkspaceBundle

Service prefix	Actions
	workspaces:DeleteWorkspaceImage
	workspaces:DeployWorkspaceApplications
	workspaces:DeregisterWorkspaceDirectory
	workspaces:DescribeAccount
	workspaces:DescribeAccountModifications
	workspaces:DescribeApplicationAssociations
	workspaces:DescribeApplications
	workspaces:DescribeBundleAssociations
	workspaces:DescribeClientBranding
	workspaces:DescribeClientProperties
	workspaces:DescribeConnectClientAddIns
	workspaces:DescribeConnectionAliasPermissions
	workspaces:DescribeConnectionAliases
	workspaces:DescribeCustomWorkspaceImageImport
	workspaces:DescribeImageAssociations
	workspaces:DescribeIpGroups
	workspaces:DescribeWorkspaceAssociations
	workspaces:DescribeWorkspaceBundles
	workspaces:DescribeWorkspaceDirectories
	workspaces:DescribeWorkspaceImagePermissions
	workspaces:DescribeWorkspaceSnapshots

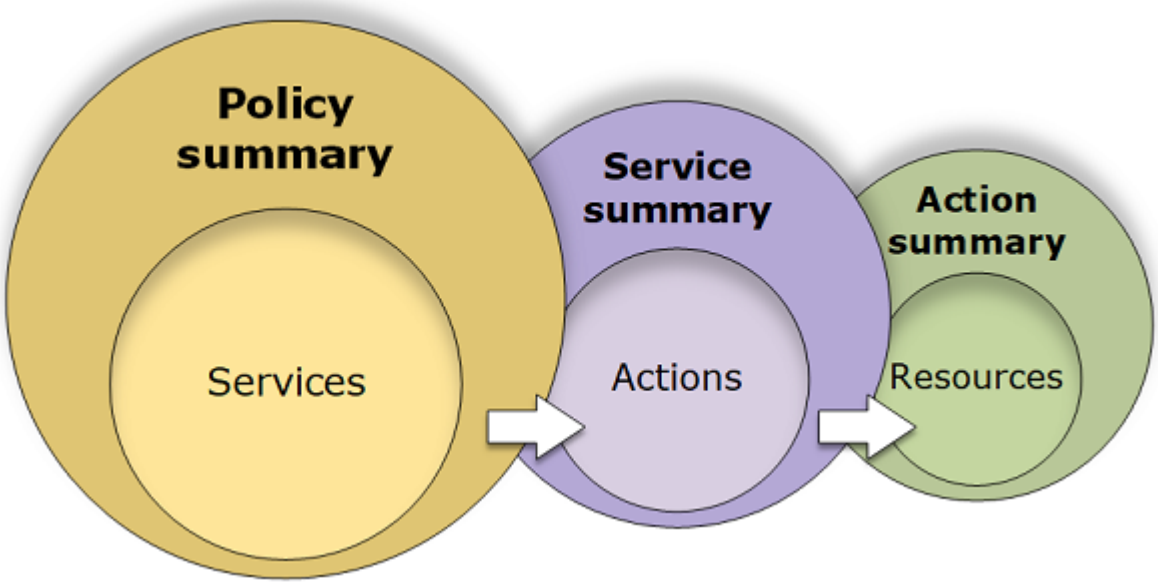
Service prefix	Actions
	workspaces:DescribeWorkspaces
	workspaces:DescribeWorkspacesConnectionStatus
	workspaces:DescribeWorkspacesPoolSessions
	workspaces:DescribeWorkspacesPools
	workspaces:DisassociateConnectionAlias
	workspaces:DisassociateIpGroups
	workspaces:DisassociateWorkspaceApplication
	workspaces:GetAccountLink
	workspaces:ImportClientBranding
	workspaces:ImportWorkspaceImage
	workspaces:ListAccountLinks
	workspaces:ListAvailableManagementCidrRanges
	workspaces:MigrateWorkspace
	workspaces:ModifyAccount
	workspaces:ModifyCertificateBasedAuthProperties
	workspaces:ModifyClientProperties
	workspaces:ModifyEndpointEncryptionMode
	workspaces:ModifySamlProperties
	workspaces:ModifySelfservicePermissions
	workspaces:ModifyStreamingProperties
	workspaces:ModifyWorkspaceAccessProperties

Service prefix	Actions
	workspaces:ModifyWorkspaceCreationProperties
	workspaces:ModifyWorkspaceProperties
	workspaces:ModifyWorkspaceState
	workspaces:RebootWorkspaces
	workspaces:RebuildWorkspaces
	workspaces:RegisterWorkspaceDirectory
	workspaces:RejectAccountLinkInvitation
	workspaces:RestoreWorkspace
	workspaces:StartWorkspaces
	workspaces:StartWorkspacesPool
	workspaces:StopWorkspaces
	workspaces:StopWorkspacesPool
	workspaces:TerminateWorkspaces
	workspaces:TerminateWorkspacesPool
	workspaces:TerminateWorkspacesPoolSession
	workspaces:UpdateConnectClientAddIn
	workspaces:UpdateConnectionAliasPermission
	workspaces:UpdateWorkspaceBundle
	workspaces:UpdateWorkspaceImagePermission
	workspaces:UpdateWorkspacesPool

Service prefix	Actions
xray	xray:CreateGroup xray:CreateSamplingRule xray:DeleteGroup xray:DeleteResourcePolicy xray:DeleteSamplingRule xray:GetEncryptionConfig xray:GetGroup xray:GetGroups xray:GetInsight xray:GetInsightEvents xray:GetInsightImpactGraph xray:GetInsightSummaries xray:GetSamplingRules xray:ListResourcePolicies xray:PutEncryptionConfig xray:PutResourcePolicy xray:UpdateGroup xray:UpdateSamplingRule

Policy summaries

The IAM console includes *policy summary* tables that describe the access level, resources, and conditions that are allowed or denied for each service in a policy. Policies are summarized in three tables: the policy summary, the service summary, and the action summary. The *policy summary* table includes a list of services. Choose a service there to see the *service summary*. This summary table includes a list of the actions and associated permissions for the chosen service. You can choose an action from that table to view the *action summary*. This table includes a list of resources and conditions for the chosen action.



You can view policy summaries on the **Users** page or **Roles** page for all policies (managed and inline) that are attached to that user. View summaries on the **Policies** page for all managed policies. Managed policies include AWS managed policies, AWS managed job function policies, and customer managed policies. You can view summaries for these policies on the **Policies** page regardless of whether they are attached to a user or other IAM identity.

You can use the information in the policy summaries to understand the permissions that are allowed or denied by your policy. Policy summaries can help you troubleshoot and fix policies that are not providing the permissions that you expect.

For step-by-step instructions on viewing summaries, see View policy summaries, View service summaries, and View action summaries.

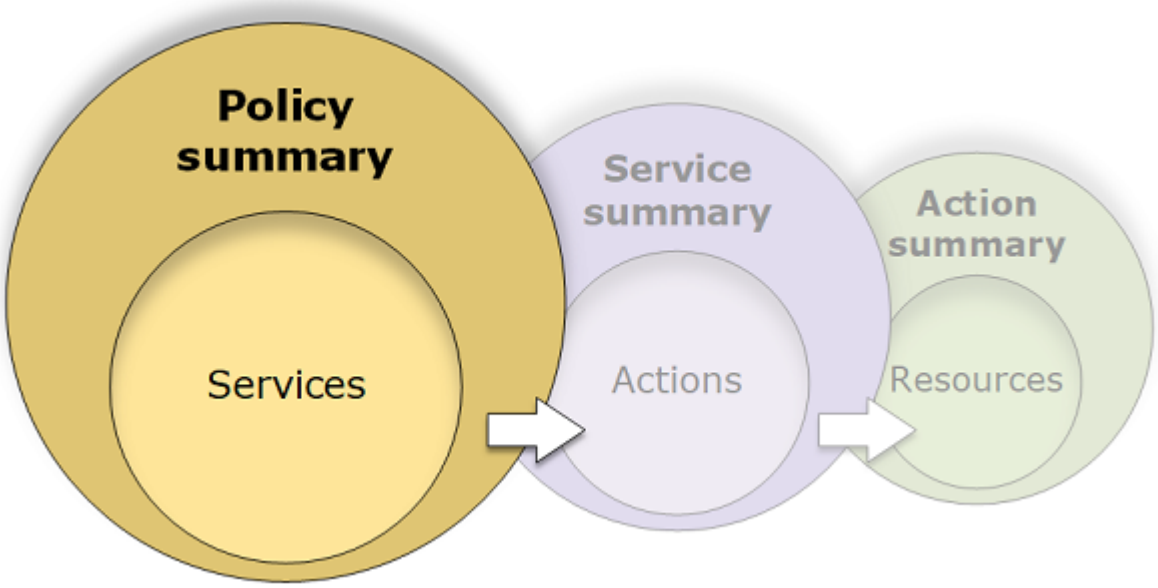
Topics

- Policy summary (list of services)

- Access levels in policy summaries
- Service summary (list of actions)
- Action summary (list of resources)
- Examples of policy summaries

Policy summary (list of services)

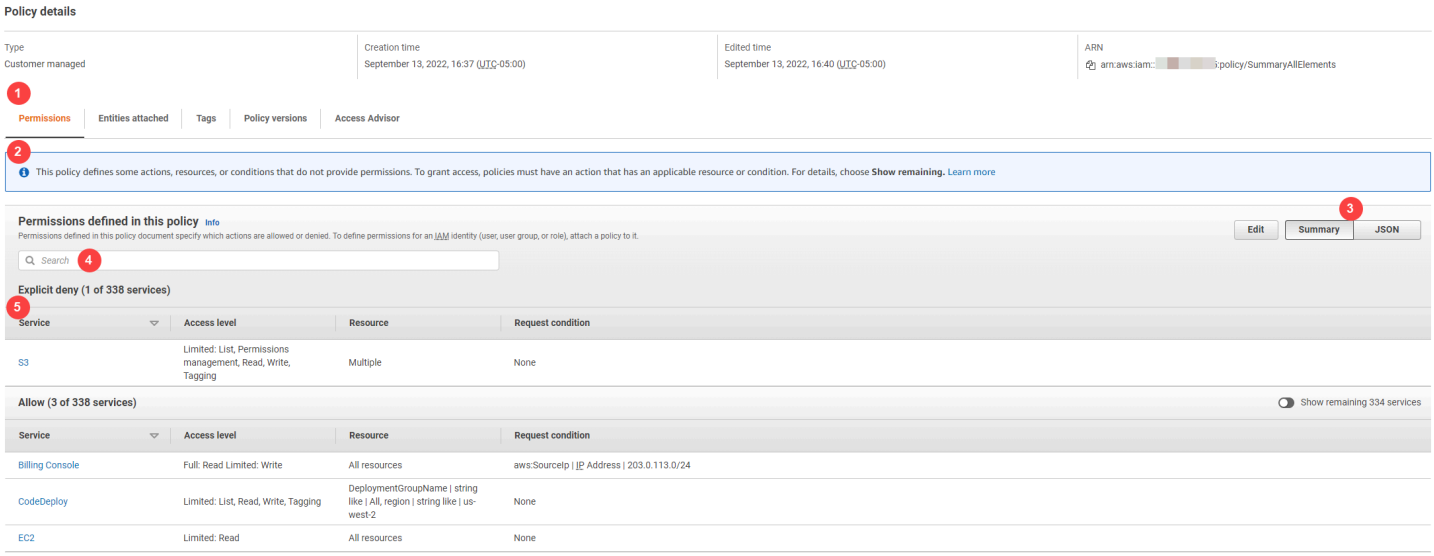
Policies are summarized in three tables: the policy summary, the service summary, and the action summary. The *policy summary* table includes a list of services and summaries of the permissions that are defined by the chosen policy.



The policy summary table is grouped into one or more **Uncategorized services**, **Explicit deny**, and **Allow** sections. If the policy includes a service that IAM does not recognize, then the service is included in the **Uncategorized services** section of the table. If IAM recognizes the service, then it is included under the **Explicit deny** or **Allow** sections of the table, depending on the effect of the policy (`Deny` or `Allow`).

Understanding the elements of a policy summary

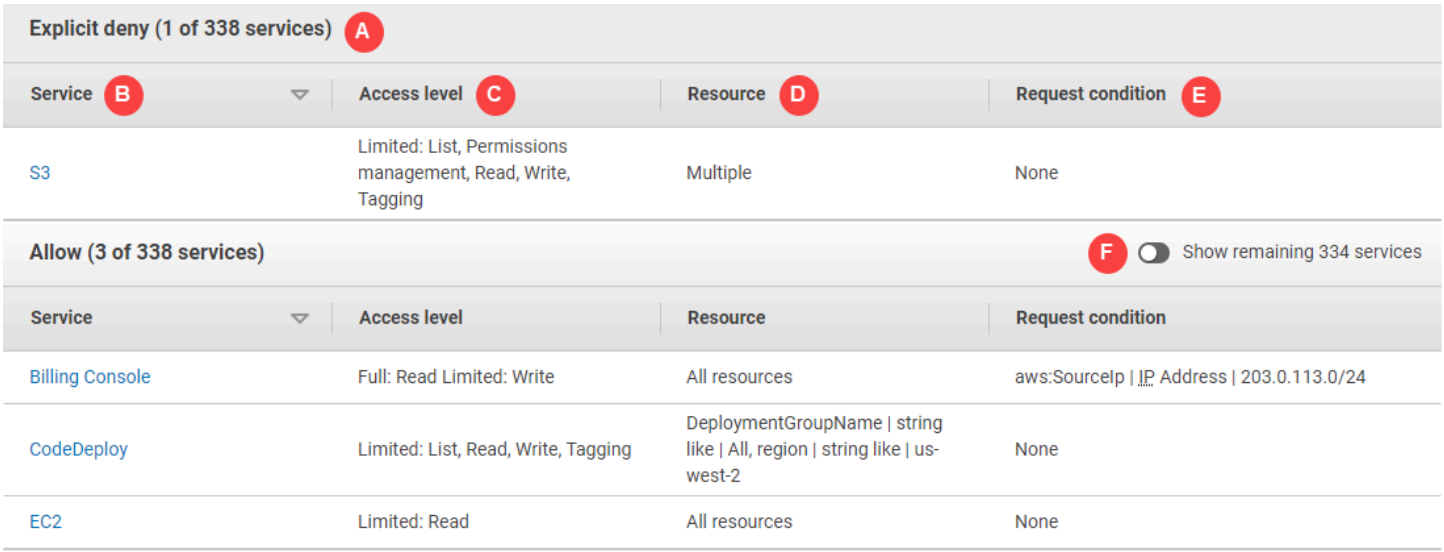
In the following example of a policy details page, the **SummaryAllElements** policy is a managed policy (customer managed policy) that is attached directly to the user. This policy is expanded to show the policy summary.



In the preceding image, the policy summary is visible from within the **Policies** page:

1. The **Permissions** tab includes the permissions defined in the policy.
2. If the policy does not grant permissions to all the actions, resources, and conditions defined in the policy, then a warning or error banner appears at the top of the page. The policy summary then includes details about the problem. To learn how policy summaries help you to understand and troubleshoot the permissions that your policy grants, see the section called “My policy does not grant the expected permissions”.
3. Use the **Summary** and **JSON** buttons to toggle between the policy summary and the JSON policy document.
4. Use the **Search** box to reduce the list of services and find a specific service.
5. The expanded view shows additional details of the **SummaryAllElements** policy.

The following policy summary table image shows the expanded **SummaryAllElements** policy on the policy details page.



Explicit deny (1 of 338 services) A

Service B	Access level C	Resource D	Request condition E
S3	Limited: List, Permissions management, Read, Write, Tagging	Multiple	None

Allow (3 of 338 services) F Show remaining 334 services

Service	Access level	Resource	Request condition
Billing Console	Full: Read Limited: Write	All resources	aws:SourceIp | IP Address | 203.0.113.0/24
CodeDeploy	Limited: List, Read, Write, Tagging	DeploymentGroupName | string like | All, region | string like | us-west-2	None
EC2	Limited: Read	All resources	None

In the preceding image, the policy summary is visible from within the **Policies** page:

A. For those services that IAM recognizes, it arranges services according to whether the policy allows or explicitly denies the use of the service. In this example, the policy includes a `Deny` statement for the Amazon S3 service and `Allow` statements for the Billing, CodeDeploy, and Amazon EC2 services.

B. **Service** – This column lists the services that are defined within the policy and provides details for each service. Each service name in the policy summary table is a link to the *service summary* table, which is explained in Service summary (list of actions). In this example, permissions are defined for the Amazon S3, Billing, CodeDeploy, and Amazon EC2 services.

C. **Access level** – This column tells whether the actions in each access level (`List`, `Read`, `Write`, `Permission Management`, and `Tagging`) have `Full` or `Limited` permissions defined in the policy. For additional details and examples of the access level summary, see Access levels in policy summaries.

- **Full access** – This entry indicates that the service has access to all actions within all four of the access levels available for the service.
- If the entry does not include **Full access**, then the service has access to some but not all of the actions for the service. The access is then defined by following descriptions for each of the access level classifications (`List`, `Read`, `Write`, `Permission Management`, and `Tagging`):

  Full: The policy provides access to all actions within each access level classification listed. In this example, the policy provides access to all of the Billing `Read` actions.

Limited: The policy provides access to one or more but not all actions within each access level classification listed. In this example, the policy provides access to some of the Billing `Write` actions.

D. **Resource** – This column shows the resources that the policy specifies for each service.

- **Multiple** – The policy includes more than one but not all of the resources within the service. In this example, access is explicitly denied to more than one Amazon S3 resource.
- **All resources** – The policy is defined for all resources within the service. In this example, the policy allows the listed actions to be performed on all Billing resources.
- Resource text – The policy includes one resource within the service. In this example, the listed actions are allowed on only the `DeploymentGroupName` CodeDeploy resource. Depending on the information that the service provides to IAM, you might see an ARN or you might see the defined resource type.

> **Note**
>
> This column can include a resource from a different service. If the policy statement that includes the resource does not include both actions and resources from the same service, then your policy includes mismatched resources. IAM does not warn you about mismatched resources when you create a policy, or when you view a policy in the policy summary. If this column includes a mismatched resource, then you should review your policy for errors. To better understand your policies, always test them with the policy simulator.

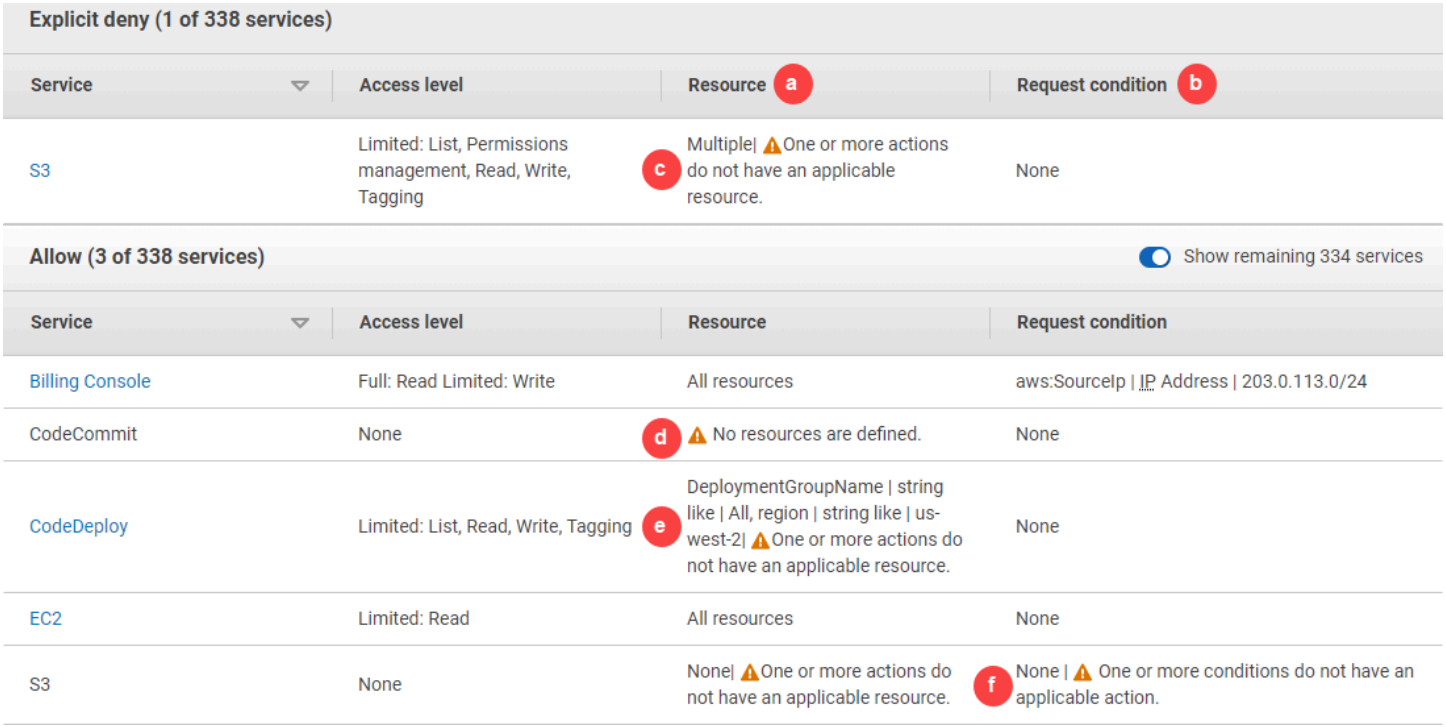
E. **Request condition** – This column indicates whether the services or actions associated with the resource are subject to conditions.

- **None** – The policy includes no conditions for the service. In this example no conditions are applied to the denied actions in the Amazon S3 service.
- Condition text – The policy includes one condition for the service. In this example, the listed Billing actions are allowed only if the IP address of the source matches `203.0.113.0/24`.
- **Multiple** – The policy includes more than one condition for the service. To view each of the multiple conditions for the policy, choose **JSON** to view the policy document.

F. **Show remaining services** – Toggle this button to expand the table to include the services that are not defined by the policy. These services are *implicitly denied* (or denied by default) within this policy. However, a statement in another policy might still allow or explicitly deny using the service. The policy summary summarizes the permissions of a single policy. To learn about

how the AWS service decides whether a given request should be allowed or denied, see Policy evaluation logic.

When a policy or an element within the policy does not grant permissions, IAM provides additional warnings and information in the policy summary. The following policy summary table shows the expanded **Show remaining services** services on the **SummaryAllElements** policy details page with the possible warnings.



Explicit deny (1 of 338 services)

Service	Access level	Resource a	Request condition b
S3	Limited: List, Permissions management, Read, Write, Tagging	c Multiple| ⚠ One or more actions do not have an applicable resource.	None

Allow (3 of 338 services) Show remaining 334 services

Service	Access level	Resource	Request condition
Billing Console	Full: Read Limited: Write	All resources	aws:SourceIp | IP Address | 203.0.113.0/24
CodeCommit	None	d ⚠ No resources are defined.	None
CodeDeploy	Limited: List, Read, Write, Tagging	e DeploymentGroupName | string like | All, region | string like | us-west-2| ⚠ One or more actions do not have an applicable resource.	None
EC2	Limited: Read	All resources	None
S3	None	None| ⚠ One or more actions do not have an applicable resource.	f None | ⚠ One or more conditions do not have an applicable action.

In the preceding image, you can see all services that include defined actions, resources, or conditions with no permissions:

a. **Resource warnings** – For services that do not provide permissions for all of the included actions or resources, you see one of the following warnings in the **Resource** column of the table:

- ⚠

  No resources are defined. – This means that the service has defined actions but no supported resources are included in the policy.

- ⚠

  One or more actions do not have an applicable resource. – This means that the service has defined actions, but that some of those actions don't have a supported resource.

- 

  One or more resources do not have an applicable action. – This means that the service has defined resources, but that some of those resources don't have a supporting action.

If a service includes both actions that do not have an applicable resource and resources that do have an applicable resource, then only the **One or more resources do not have an applicable action.** warning is shown. This is because when you view the service summary for the service, resources that do not apply to any action are not shown. For the `ListAllMyBuckets` action, this policy includes the last warning because the action does not support resource-level permissions, and does not support the `s3:x-amz-acl` condition key. If you fix either the resource problem or the condition problem, the remaining issue appears in a detailed warning.

b. **Request condition warnings** – For services that do not provide permissions for all of the included conditions, you see one of the following warnings in the **Request condition** column of the table:

- 

  One or more actions do not have an applicable condition. – This means that the service has defined actions, but that some of those actions don't have a supported condition.

- 

  One or more conditions do not have an applicable action. – This means that the service has defined conditions, but that some of those conditions don't have a supporting action.

c. **Multiple |**



One or more actions do not have an applicable resource. – The `Deny` statement for Amazon S3 includes more than one resource. It also includes more than one action, and some actions support the resources and some do not. To view this policy, see the section called "**SummaryAllElements** JSON policy document". In this case, the policy includes all Amazon S3 actions, and only the actions that can be performed on a bucket or bucket object are denied.

d. 

No resources are defined – The service has defined actions, but no supported resources are included in the policy, and therefore the service provides no permissions. In this case, the policy includes CodeCommit actions but no CodeCommit resources.

e. **DeploymentGroupName | string like | All, region | string like | us-west-2 |**



One or more actions do not have an applicable resource. – The service has a defined action, and at least one more action that does not have a supporting resource.

f. **None |**



One or more conditions do not have an applicable action. – The service has at least one condition key that does not have a supporting action.

SummaryAllElements JSON policy document

The **SummaryAllElements** policy is not intended for you to use to define permissions in your account. Rather, it is included to demonstrate the errors and warnings that you might encounter while viewing a policy summary.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "billing:Get*",
                "payments:List*",
                "payments:Update*",
                "account:Get*",
                "account:List*",
                "cur:GetUsage*"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": "203.0.113.0/24"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
```

```
                "s3:*"
            ],
            "Resource": [
                "arn:aws:s3:::customer",
                "arn:aws:s3:::customer/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:GetConsoleScreenshots"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "codedploy:*",
                "codecommit:*"
            ],
            "Resource": [
                "arn:aws:codedeploy:us-west-2:123456789012:deploymentgroup:*",
                "arn:aws:codebuild:us-east-1:123456789012:project/my-demo-
project"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:GetObject",
                "s3:DeletObject",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:autoscling:us-east-2:123456789012:autoscalgrp"
            ],
            "Condition": {
                "StringEquals": {
```

```
                    "s3:x-amz-acl": [
                        "public-read"
                    ],
                    "s3:prefix": [
                        "custom",
                        "other"
                    ]
                }
            }
        }
    ]
}
```

View policy summaries

You can view the policy summaries for any policies that are attached to an IAM user or role. For managed policies, you can view policy summaries on the **Policies** page. If your policy does not include a policy summary, see Missing policy summary to learn why.

Viewing policy summaries from the Policies page

You can view the policy summary for managed policies on the **Policies** page.

To view the policy summary from the Policies page

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to view.
4. On the **Policy details** page for the policy, view the **Permissions** tab to see the policy summary.

Viewing a policy summary for a policy attached to a user

You can view the policy summary for any policies that are attached to an IAM user.

To view the summary for a policy attached to a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. Choose **Users** from the navigation pane.
3. In the list of users, choose the name of the user whose policy you want to view.
4. On the **Summary** page for the user, view the **Permissions** tab to see the list of policies that are attached to the user directly or from a group.
5. In the table of policies for the user, expand the row of the policy that you want to view.

Viewing a policy summary for a policy attached to a role

You can view the policy summary for any policies that are attached to a role.

To view the summary for a policy attached to a role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. In the list of roles, choose the name of the role whose policy you want to view.
4. On the **Summary** page for the role, view the **Permissions** tab to see the list of policies that are attached to the role.
5. In the table of policies for the role, expand the row of the policy that you want to view.

Editing policies to fix warnings

While viewing a policy summary, you might find a typo or notice that the policy does not provide the permissions that you expected. You cannot edit a policy summary directly. However, you can edit a customer managed policy using the visual policy editor, which catches many of the same errors and warnings that the policy summary reports. You can then view the changes in the policy summary to confirm that you fixed all of the issues. To learn how to edit an inline policy, see the section called “Edit IAM policies”. You cannot edit AWS managed policies.

You can edit a policy for your policy summary using the **Visual** option.

To edit a policy for your policy summary using the Visual option

1. Open the policy summary as explained in the previous procedures.
2. Choose **Edit**.

If you are on the **Users** page and choose to edit a customer managed policy that is attached to that user, you are redirected to the **Policies** page. You can edit customer managed policies only on the **Policies** page.

3. Choose the **Visual** option to view the editable visual representation of your policy. IAM might restructure your policy to optimize it for the visual editor and to make it easier for you to find and fix any problems. The warnings and error messages on the page can guide you to fix any issues with your policy. For more information about how IAM restructures policies, see Policy restructuring.
4. Edit your policy and choose **Next** to see your changes reflected in the policy summary. If you still see a problem, choose **Previous** to return to the editing screen.
5. Choose **Save changes** to save your changes.

You can edit a policy for your policy summary using the **JSON** option.

To edit a policy for your policy summary using the JSON option

1. Open the policy summary as explained in the previous procedures.
2. You can use the **Summary** and **JSON** buttons to compare the policy summary to the JSON policy document. You can use this information to determine which lines in the policy document you want to change.
3. Choose **Edit** and then choose the **JSON** option to edit the JSON policy document.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor option, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

If you are on the **Users** page and choose to edit a customer managed policy that is attached to that user, you are redirected to the **Policies** page. You can edit customer managed policies only on the **Policies** page.

4. Edit your policy. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**. If you still see a problem, choose **Previous** to return to the editing screen.
5. Choose **Save changes** to save your changes.

Access levels in policy summaries

AWS access level summary

Policy summaries include an access level summary that describes the action permissions defined for each service that is mentioned in the policy. To learn about policy summaries, see Policy summaries. Access level summaries indicate whether the actions in each access level (`List`, `Read`, `Tagging`, `Write`, and `Permissions management`) have `Full` or `Limited` permissions defined in the policy. To view the access level classification that is assigned to each action in a service, see Actions, Resources, and Condition Keys for AWS Services.

The following example describes the access provided by a policy for the given services. For examples of full JSON policy documents and their related summaries, see Examples of policy summaries.

Service	Access level	This policy provides the following
IAM	Full access	Access to all actions within the IAM service.
CloudWatch	**Full:** List	Access to all CloudWatch actions in the `List` access level, but no access to actions with the `Read`, `Write`, or `Permissions management` access level classification.
Data Pipeline	**Limited:** List, Read	Access to at least one but not all AWS Data Pipeline actions in the `List` and `Read` access level, but not the `Write` or `Permissions management` actions.
EC2	**Full:** List, Read **Limited:** Write	Access to all Amazon EC2 `List` and `Read` actions and access to at least one but not all Amazon EC2 `Write` actions, but no access to

Service	Access level	This policy provides the following
		actions with the `Permissions management` access level classification.
S3	**Limited:** Read, Write, Permissions management	Access to at least one but not all Amazon S3 `Read`, `Write` and `Permissions management` actions.
codedploy	(empty)	Unknown access, because IAM does not recognize this service.
API Gateway	None	No access is defined in the policy.
CodeBuild	⚠ No actions are defined.	No access because no actions are defined for the service. To learn how to understand and troubleshoot this issue, see the section called "My policy does not grant the expected permissions".

In a policy summary, **Full access** indicates that the policy provides access to all the actions within the service. Policies that provide access to some but not all actions within a service are further grouped according to the access level classification. This is indicated by one of the following access-level groupings:

- **Full**: The policy provides access to all actions within the specified access level classification.
- **Limited**: The policy provides access to one or more but not all actions within the specified access level classification.
- **None**: The policy provides no access.
- (empty): IAM does not recognize this service. If the service name includes a typo, then the policy provides no access to the service. If the service name is correct, then the service might not support policy summaries or might be in preview. In this case, the policy might provide access, but that access cannot be shown in the policy summary. To request policy summary support for a generally available (GA) service, see Service does not support IAM policy summaries.

Access level summaries that include limited (partial) access to actions are grouped using the AWS access level classifications `List`, `Read`, `Tagging`, `Write`, or `Permissions management`.

AWS access levels

AWS defines the following access level classifications for the actions in a service:

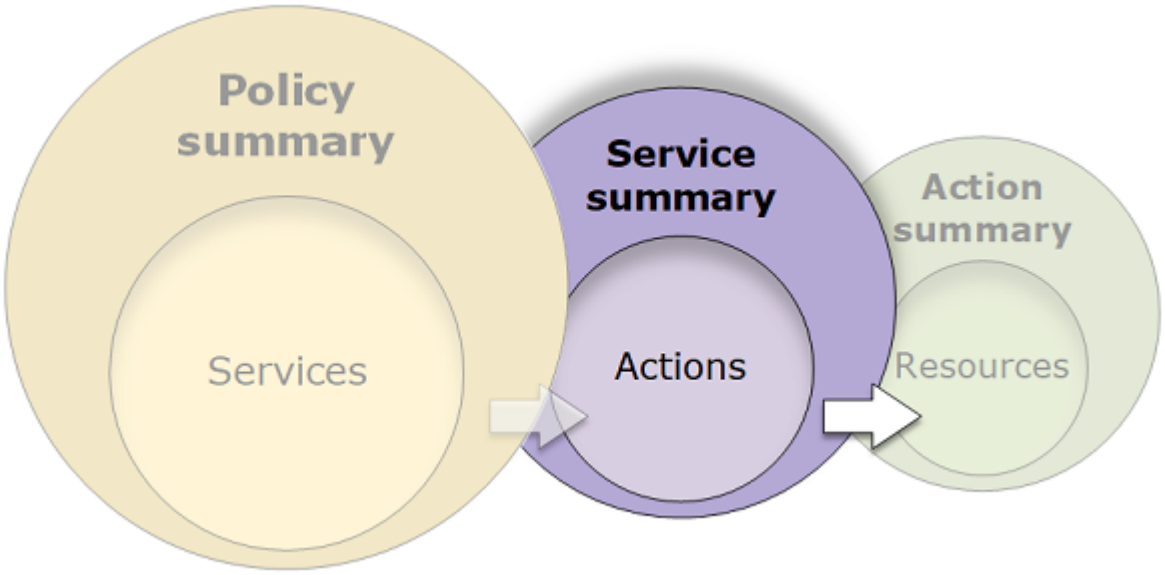
- **List**: Permission to list resources within the service to determine whether an object exists. Actions with this level of access can list objects but cannot see the contents of a resource. For example, the Amazon S3 action `ListBucket` has the **List** access level.
- **Read**: Permission to read but not edit the contents and attributes of resources in the service. For example, the Amazon S3 actions `GetObject` and `GetBucketLocation` have the **Read** access level.
- **Tagging**: Permission to perform actions that only change the state of resource tags. For example, the IAM actions `TagRole` and `UntagRole` have the **Tagging** access level because they allow only tagging or untagging a role. However, the `CreateRole` action allows tagging a role resource when you create that role. Because the action does not only add a tag, it has the `Write` access level.
- **Write**: Permission to create, delete, or modify resources in the service. For example, the Amazon S3 actions `CreateBucket`, `DeleteBucket` and `PutObject` have the **Write** access level. `Write` actions might also allow modifying a resource tag. However, an action that allows only changes to tags has the `Tagging` access level.
- **Permissions management**: Permissions management refers to actions that control access within AWS services, including IAM and non-IAM identity permissions, but excludes network-level access controls like security groups. For example, most IAM and AWS Organizations actions, as well as the Amazon S3 actions `PutBucketPolicy` and `DeleteBucketPolicy` have the **Permissions management** access level.

> **Tip**
>
> To improve the security of your AWS account, restrict or regularly monitor policies that include the **Permissions management** access level classification.

To view the access level classification for all of the actions in a service, see Actions, Resources, and Condition Keys for AWS Services.

Service summary (list of actions)

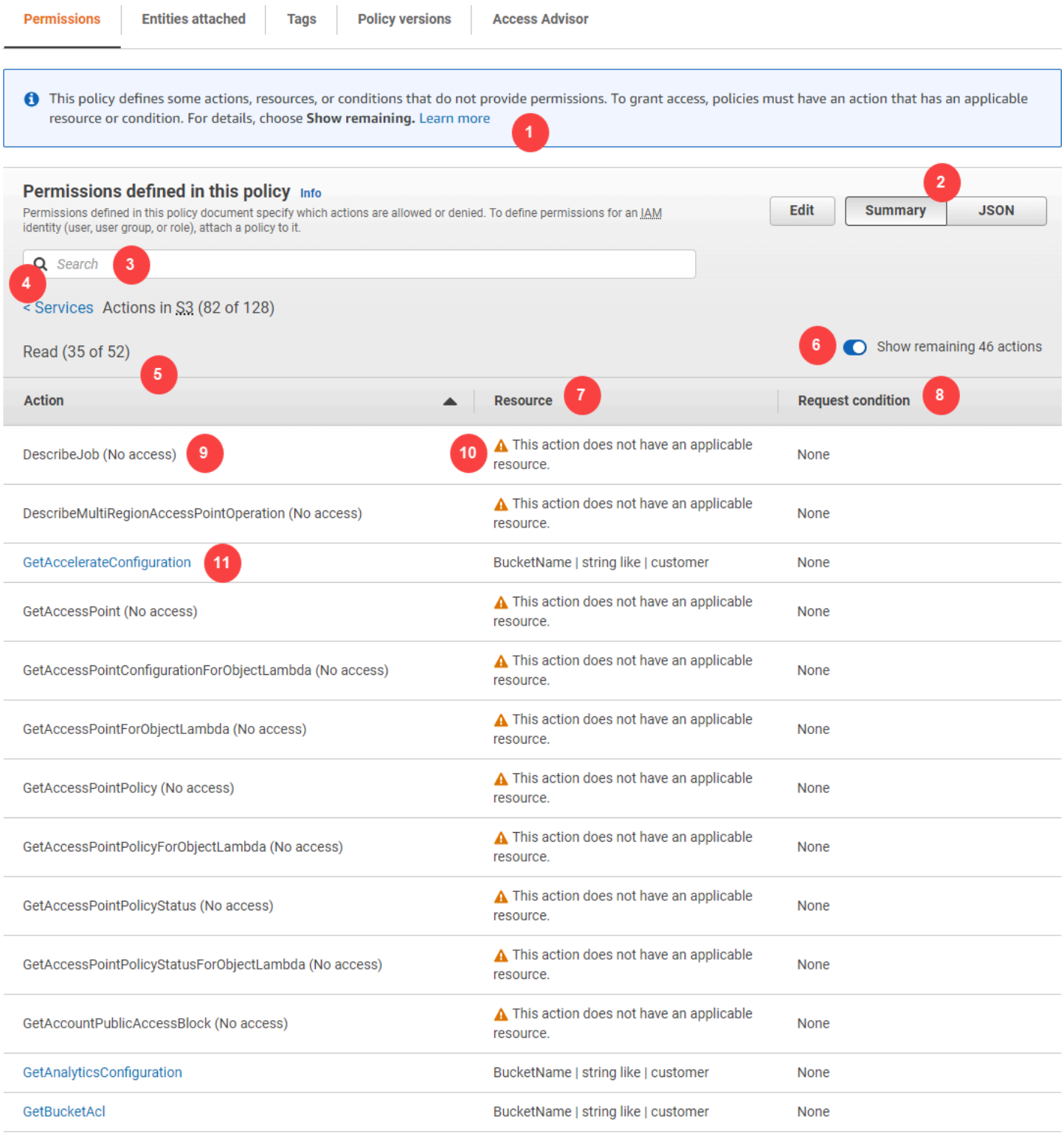
Policies are summarized in three tables: the policy summary, the service summary, and the action summary. The *service summary* table includes a list of the actions and summaries of the permissions that are defined by the policy for the chosen service.



You can view a service summary for each service listed in the policy summary that grants permissions. The table is grouped into **Uncategorized actions**, **Uncategorized resource types**, and access level sections. If the policy includes an action that IAM does not recognize, then the action is included in the **Uncategorized actions** section of the table. If IAM recognizes the action, then it is included under one of the access level (**List**, **Read**, **Write** and **Permissions management**) sections of the table. To view the access level classification that is assigned to each action in a service, see Actions, Resources, and Condition Keys for AWS Services.

Understanding the elements of a service summary

The example below is the service summary for Amazon S3 actions that are allowed from a policy summary. The actions for this service are grouped by access level. For example, 35 **Read** actions are defined out of the total 52 **Read** actions available for the service.



The service summary page for a managed policy includes the following information:

1. If the policy does not grant permissions to all the actions, resources, and conditions defined for the service in the policy, then a warning banner appears at the top of the page. The service

summary then includes details about the problem. To learn how policy summaries help you to understand and troubleshoot the permissions that your policy grants, see the section called "My policy does not grant the expected permissions".

2. Choose **JSON** to see additional details about the policy. You can do this to view all conditions that are applied to the actions. (If you are viewing the service summary for an inline policy that is attached directly to a user, you must close the service summary dialog box and return to the policy summary to access the JSON policy document.)
3. To view the summary for a specific action, type keywords into the **Search** box to reduce the list of available actions.
4. Next to the **Services** back arrow appears the name of the service (in this case **S3**). The service summary for this service includes the list of allowed or denied actions that are defined in the policy. If the service appears under **(Explicit deny)** on the **Permissions** tab, then the actions listed in the service summary table are explicitly denied. If the service appears under **Allow** on the **Permissions** tab, then the actions listed in the service summary table are allowed.
5. **Action** – This column lists the actions that are defined within the policy and provides the resources and conditions for each action. If the policy grants or denies permissions to the action, then the action name links to the *action summary* table. The table groups these actions into at least one or up to five sections, depending on the level of access that the policy allows or denies. The sections are **List**, **Read**, **Write**, **Permission Management**, and **Tagging**. The count indicates the number of recognized actions that provide permissions within each access level. The total is the number of known actions for the service. In this example, 35 actions provide permissions out of 52 total known Amazon S3 **Read** actions. To view the access level classification that is assigned to each action in a service, see Actions, Resources, and Condition Keys for AWS Services.
6. **Show remaining actions** – Toggle this button to expand or hide the table to include actions that are known but do not provide permissions for this service. Toggling the button also displays warnings for any elements that do not provide permissions.
7. **Resource** – This column shows the resources that the policy defines for the service. IAM does not check whether the resource applies to each action. In this example, actions in the Amazon S3 service are allowed on only the `developer_bucket` Amazon S3 bucket resource. Depending on the information that the service provides to IAM, you might see an ARN such as `arn:aws:s3:::developer_bucket/*`, or you might see the defined resource type, such as `BucketName = developer_bucket`.

> **Note**
>
> This column can include a resource from a different service. If the policy statement that includes the resource does not include both actions and resources from the same service, then your policy includes mismatched resources. IAM does not warn you about mismatched resources when you create a policy, or when you view a policy in the service summary. IAM also does not indicate whether the action applies to the resources, only whether the service matches. If this column includes a mismatched resource, then you should review your policy for errors. To better understand your policies, always test them with the policy simulator.

8. **Request condition** – This column tells whether the actions associated with the resource are subject to conditions. To learn more about those conditions, choose **JSON** to review the JSON policy document.
9. **(No access)** – This policy includes an action that does not provide permissions.
10. **Resource warning** – For actions with resources that do not provide full permissions, you see one of the following warnings:
    - **This action does not support resource-level permissions. This requires a wildcard (*) for the resource.** – This means that the policy includes resource-level permissions but must include `"Resource": ["*"]` to provide permissions for this action.
    - **This action does not have an applicable resource.** – This means that the action is included in the policy without a supported resource.
    - **This action does not have an applicable resource and condition.** – This means that the action is included in the policy without a supported resource and without a supported condition. In this case, there is also condition included in the policy for this service, but there are no conditions that apply to this action.
11. Actions that provide permissions include a link to the action summary.

View service summaries

You can view a service summary for each service listed in the policy summary that grants permissions.

Viewing service summaries from the Policies page

You can view the service summary for managed policies on the **Policies** page.

To view the service summary for a managed policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to view.
4. On the **Policy details** page for the policy, view the **Permissions** tab to see the policy summary.
5. In the policy summary list of services, choose the name of the service that you want to view.

Viewing a service summary for a policy attached to a user

You can view the service summary for any policies that are attached to an IAM user.

To view the service summary for a policy attached to a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. In the list of users, choose the name of the user whose policy you want to view.
4. On the **Summary** page for the user, view the **Permissions** tab to see the list of policies that are attached to the user directly or from a group.
5. In the table of policies for the user, choose the name of the policy that you want to view.

   If you are on the **Users** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.
6. Choose **Summary**. In the policy summary list of services, choose the name of the service that you want to view.

> **Note**
>
> If the policy that you select is an inline policy that is attached directly to the user, then the service summary table appears. If the policy is an inline policy attached from a group, then you are taken to the JSON policy document for that group. If the policy is a managed policy, then you are taken to the service summary for that policy on the **Policies** page.

Viewing a service summary for a policy attached to a role

You can view the policy summary for any policies that are attached to a role.

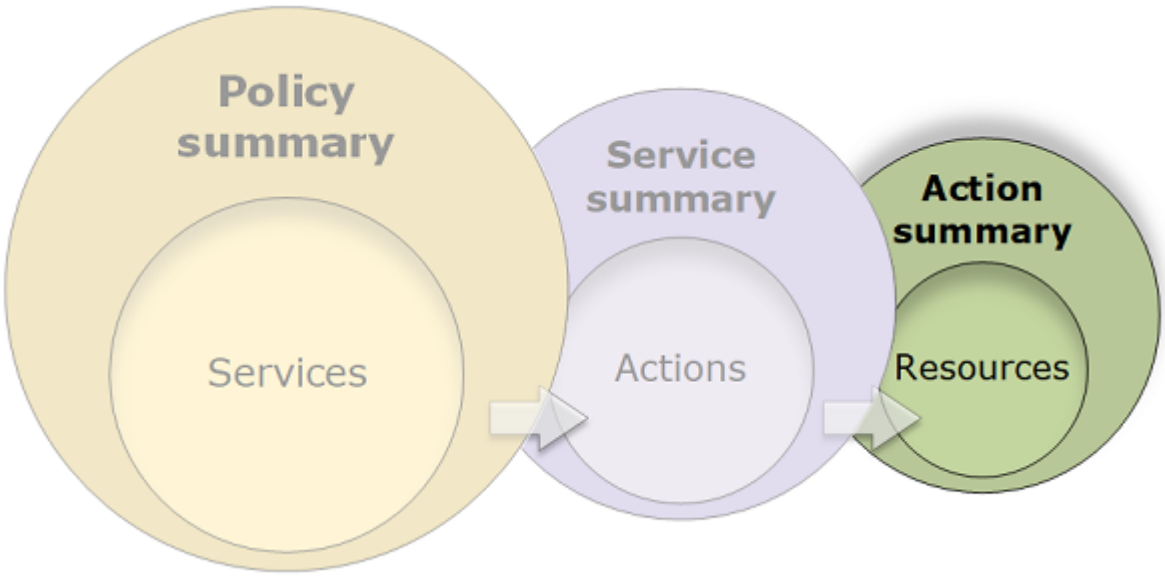
To view the service summary for a policy attached to a role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles** from the navigation pane.
3. In the list of roles, choose the name of the role whose policy you want to view.
4. On the **Summary** page for the role, view the **Permissions** tab to see the list of policies that are attached to the role.
5. In the table of policies for the role, choose the name of the policy that you want to view.

   If you are on the **Roles** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.
6. In the policy summary list of services, choose the name of the service that you want to view.

Action summary (list of resources)

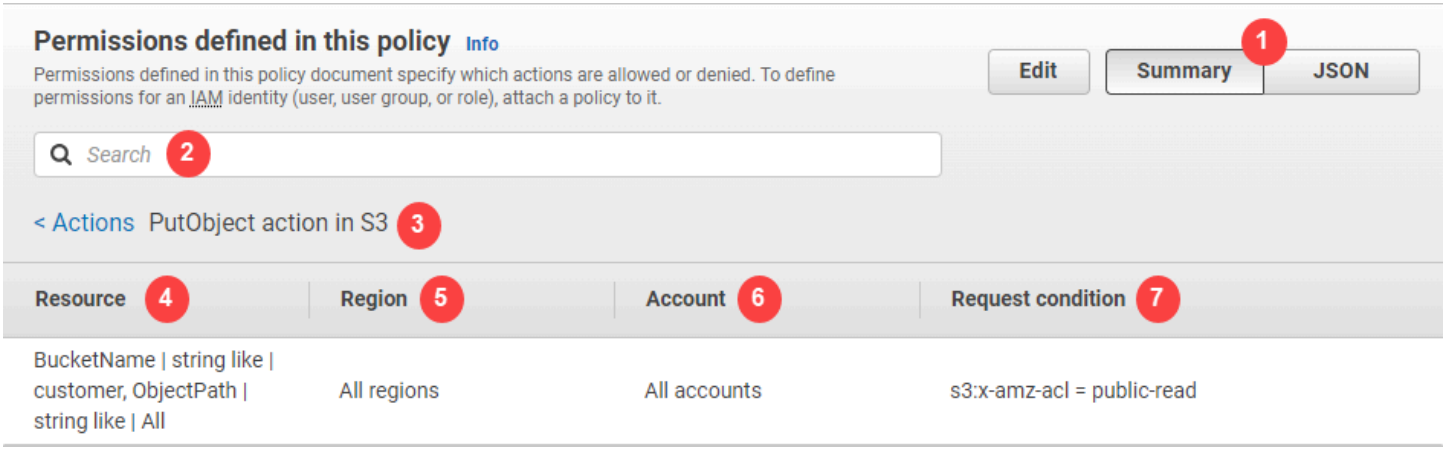
Policies are summarized in three tables: the policy summary, the service summary, and the action summary. The *action summary* table includes a list of resources and the associated conditions that apply to the chosen action.



To view an action summary for each action that grants permissions, choose the link in the service summary. The action summary table includes details about the resource, including its **Region** and **Account**. You can also view the conditions that apply to each resource. This shows you conditions that apply to some resources but not others.

Understanding the elements of an action summary

The example below is the action summary for the `PutObject` (Write) action from the Amazon S3 service summary (see Service summary (list of actions)). For this action, the policy defines multiple conditions on a single resource.



The action summary page includes the following information:

1. Choose **JSON** to see additional details about the policy, such as viewing the multiple conditions that are applied to the actions. (If you are viewing the action summary for an inline policy that is attached directly to a user, the steps differ. To access the JSON policy document in that case, you must close the action summary dialog box and return to the policy summary.)
2. To view the summary for a specific resource, type keywords into the **Search** box to reduce the list of available resources.
3. Next to the **Actions** back arrow appears the name of the service and action in the format `action name action in service` (in this case **PutObject action in S3**). The action summary for this service includes the list of resources that are defined in the policy.
4. **Resource** – This column lists the resources that the policy defines for the chosen service. In this example, the **PutObject** action is allowed on all object paths, but on only the `developer_bucket` Amazon S3 bucket resource. Depending on the information that the service provides to IAM, you might see an ARN such as `arn:aws:s3:::developer_bucket/`

*, or you might see the defined resource type, such as `BucketName = developer_bucket, ObjectPath = All`.

5. **Region** – This column shows the Region in which the resource is defined. Resources can be defined for all Regions, or a single Region. They cannot exist in more than one specific Region.
   - **All regions** – The actions that are associated with the resource apply to all Regions. In this example, the action belongs to a global service, Amazon S3. Actions that belong to global services apply to all Regions.
   - Region text – The actions associated with the resource apply to one Region. For example, a policy can specify the `us-east-2` Region for a resource.
6. **Account** – This column indicates whether the services or actions associated with the resource apply to a specific account. Resources can exist in all accounts or a single account. They cannot exist in more than one specific account.
   - **All accounts** – The actions that are associated with the resource apply to all accounts. In this example, the action belongs to a global service, Amazon S3. Actions that belong to global services apply to all accounts.
   - **This account** – The actions that are associated with the resource apply only in the current account..
   - Account number – The actions that are associated with the resource apply to one account (one that you are not currently logged in to). For example, if a policy specifies the `123456789012` account for a resource, then the account number appears in the policy summary.
7. **Request condition** – This column shows whether the actions that are associated with the resource are subject to conditions. This example includes the `s3:x-amz-acl = public-read` condition. To learn more about those conditions, choose **JSON** to review the JSON policy document.

View action summaries

You can view an action summary for each action listed in the policy summary that grants permissions.

Viewing action summaries from the Policies page

You can view the action summary for managed policies.

To view the action summary for a managed policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to view.
4. On the **Policy details** page for the policy, view the **Permissions** tab to see the policy summary.
5. In the policy summary list of services, choose the name of the service that you want to view.
6. In the service summary list of actions, choose the name of the action that you want to view.

Viewing action summaries for a policy attached to a user

You can view the action summary for any policy that is attached to a user.

To view the action summary for a policy attached to a user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Users** from the navigation pane.
3. In the list of users, choose the name of the user whose policy you want to view.
4. On the **Summary** page for the user, view the **Permissions** tab to see the list of policies that are attached to the user directly or from a group.
5. In the table of policies for the user, choose the name of the policy that you want to view.

   If you are on the **Users** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.
6. In the policy summary list of services, choose the name of the service that you want to view.

> **Note**
>
> If the policy that you select is an inline policy that is attached directly to the user, then the service summary table appears. If the policy is an inline policy attached from a group, then you are taken to the JSON policy document for that group. If the policy is a managed policy, then you are taken to the service summary for that policy on the **Policies** page.

7. In the service summary list of actions, choose the name of the action that you want to view.

Viewing action summaries for a policy attached to a role

You can view the action summary for any policy that is attached to a role.

To view the action summary for a policy attached to a role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. In the list of roles, choose the name of the role whose policy you want to view.
4. On the **Summary** page for the role, view the **Permissions** tab to see the list of policies that are attached to the role.
5. In the table of policies for the role, choose the name of the policy that you want to view.

   If you are on the **Roles** page and choose to view the service summary for a policy that is attached to that user, you are redirected to the **Policies** page. You can view service summaries only on the **Policies** page.
6. In the policy summary list of services, choose the name of the service that you want to view.
7. In the service summary list of actions, choose the name of the action that you want to view.

Examples of policy summaries

The following examples include JSON policies with their associated policy summaries, the service summaries, and the action summaries to help you understand the permissions given through a policy.

Policy 1: DenyCustomerBucket

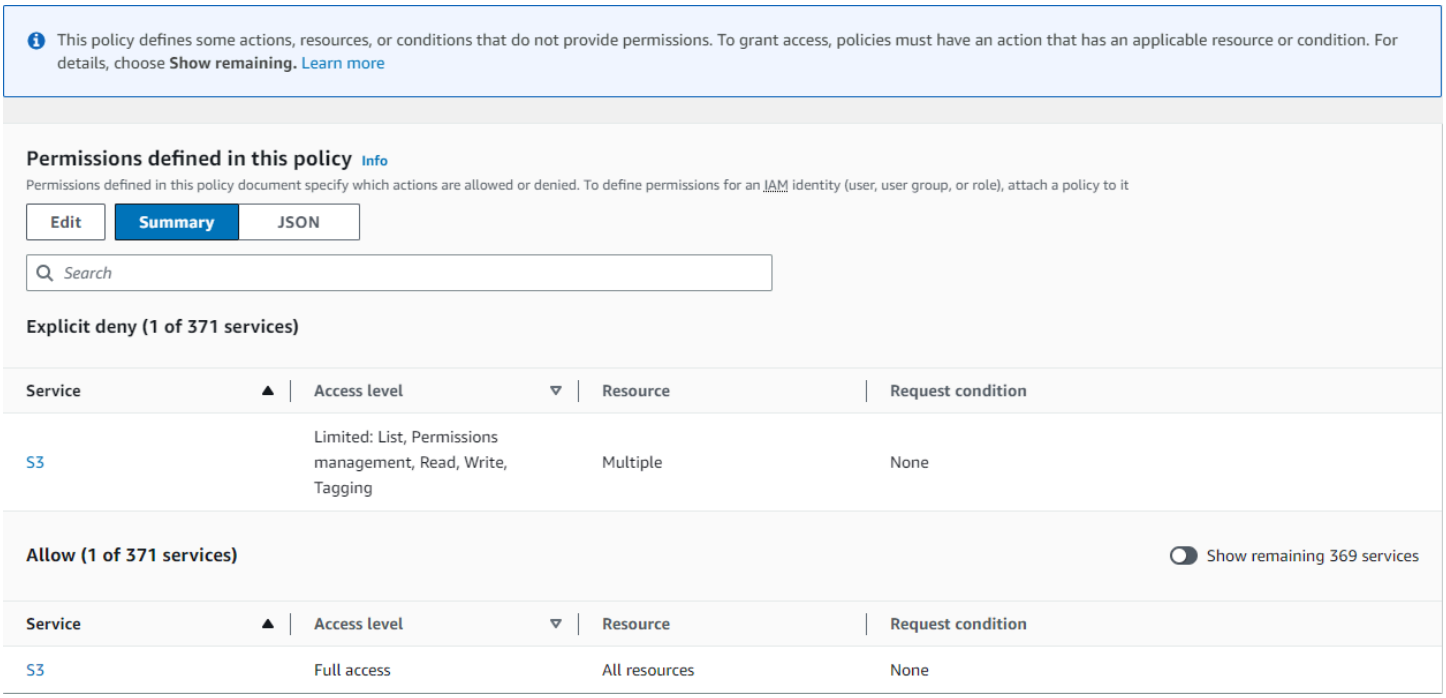
This policy demonstrates an allow and a deny for the same service.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "FullAccess",
            "Effect": "Allow",
            "Action": ["s3:*"],
            "Resource": ["*"]
        },
        {
            "Sid": "DenyCustomerBucket",
            "Action": ["s3:*"],
            "Effect": "Deny",
            "Resource": ["arn:aws:s3:::customer", "arn:aws:s3:::customer/*" ]
        }
    ]
}
```

***DenyCustomerBucket** Policy Summary:*



***DenyCustomerBucket S3 (Explicit deny)** Service Summary:*

< Services Actions in S3 (82 of 130) Show remaining 48 actions

Read (35 of 53)

Action	Resource	Request condition
GetAccelerateConfiguration	BucketName| string like |customer	None
GetAnalyticsConfiguration	BucketName| string like |customer	None
GetBucketAcl	BucketName| string like |customer	None
GetBucketCORS	BucketName| string like |customer	None
GetBucketLocation	BucketName| string like |customer	None
GetBucketLogging	BucketName| string like |customer	None
GetBucketNotification	BucketName| string like |customer	None
GetBucketObjectLockConfiguration	BucketName| string like |customer	None
GetBucketOwnershipControls	BucketName| string like |customer	None
GetBucketPolicy	BucketName| string like |customer	None
GetBucketPolicyStatus	BucketName| string like |customer	None
GetBucketPublicAccessBlock	BucketName| string like |customer	None
GetBucketRequestPayment	BucketName| string like |customer	None
GetBucketTagging	BucketName| string like |customer	None
GetBucketVersioning	BucketName| string like |customer	None
GetBucketWebsite	BucketName| string like |customer	None

***GetObject (Read)** Action Summary:*

< Actions GetObject action in S3

Resource	Region	Account	Request condition
BucketName| string like |customer, ObjectPath| string like |All	-	All accounts	None

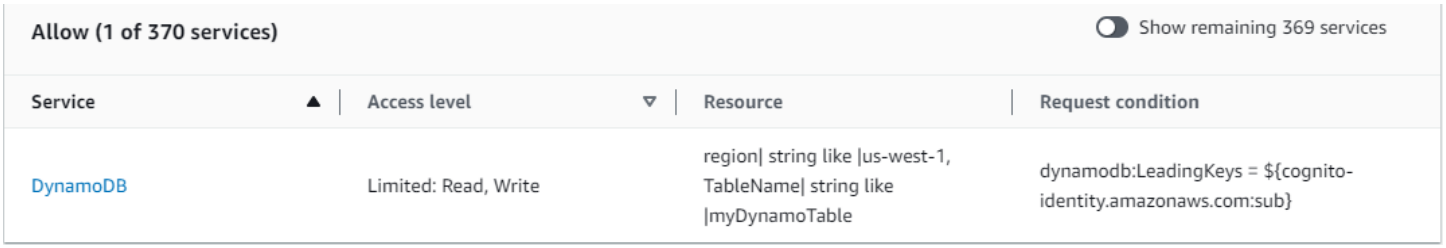
Policy 2: DynamoDbRowCognitoID

This policy provides row-level access to Amazon DynamoDB based on the user's Amazon Cognito ID.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-1:123456789012:table/myDynamoTable"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": [
                        "${cognito-identity.amazonaws.com:sub}"
                    ]
                }
            }
        }
    ]
}
```

DynamoDbRowCognitoID *Policy Summary:*



DynamoDbRowCognitoID DynamoDB (Allow) *Service Summary:*

< Services Actions in DynamoDB (4 of 65) Show remaining 61 actions

Read (1 of 26)

Action	Resource	Request condition
GetItem	region| string like |us-west-1, TableName| string like |myDynamoTable	dynamodb:LeadingKeys = ${cognito-identity.amazonaws.com:sub}

Write (3 of 33)

Action	Resource	Request condition
DeleteItem	region| string like |us-west-1, TableName| string like |myDynamoTable	dynamodb:LeadingKeys = ${cognito-identity.amazonaws.com:sub}
PutItem	region| string like |us-west-1, TableName| string like |myDynamoTable	dynamodb:LeadingKeys = ${cognito-identity.amazonaws.com:sub}
UpdateItem	region| string like |us-west-1, TableName| string like |myDynamoTable	dynamodb:LeadingKeys = ${cognito-identity.amazonaws.com:sub}

***GetItem (List)** Action Summary:*

< Actions GetItem action in DynamoDB

Resource	Region	Account	Request condition
region| string like |us-west-1, TableName| string like |myDynamoTable	us-west-1	123456789012	dynamodb:LeadingKeys = ${cognito-identity.amazonaws.com:sub}

Policy 3: MultipleResourceCondition

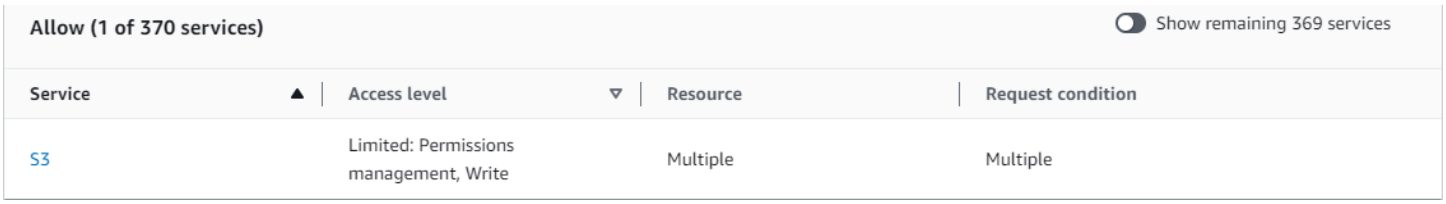
This policy includes multiple resources and conditions.

JSON

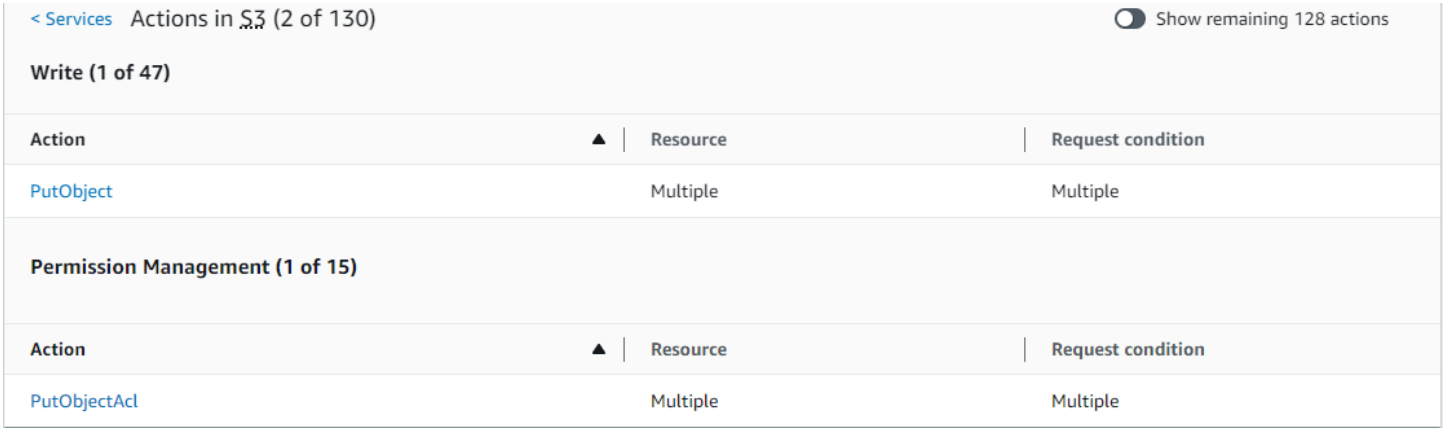
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": ["arn:aws:s3:::Apple_bucket/*"],
            "Condition": {"StringEquals": {"s3:x-amz-acl": ["public-read"]}}
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": ["arn:aws:s3:::Orange_bucket/*"],
            "Condition": {"StringEquals": {
                "s3:x-amz-acl": ["custom"],
                "s3:x-amz-grant-full-control": ["1234"]
            }}
        }
    ]
}
```

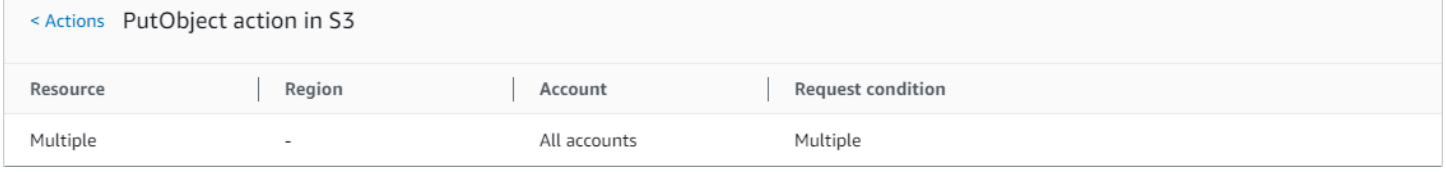
***MultipleResourceCondition** Policy Summary:*



***MultipleResourceCondition S3 (Allow)** Service Summary:*



***PutObject (Write)** Action Summary:*



Policy 4: EC2_troubleshoot

The following policy allows users to get a screenshot of a running Amazon EC2 instance, which can help with EC2 troubleshooting. This policy also permits viewing information about the items in the Amazon S3 developer bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:GetConsoleScreenshot"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::developer"
            ]
        }
    ]
}
```

***EC2_Troubleshoot** Policy Summary:*

Allow (2 of 370 services) Show remaining 368 services

Service	Access level	Resource	Request condition
EC2	Limited: Read	All resources	None
S3	Limited: List	BucketName| string like |developer	None

***EC2_Troubleshoot S3 (Allow)** Service Summary:*

< Services Actions in S3 (1 of 130) Show remaining 129 actions

List (1 of 7)

Action	Resource	Request condition
ListBucket	BucketName| string like |developer	None

***ListBucket (List)** Action Summary:*

< Actions ListBucket action in S3

Resource	Region	Account	Request condition
BucketName| string like |developer	-	All accounts	None

Policy 5: CodeBuild_CodeCommit_CodeDeploy

This policy provides access to specific CodeBuild, CodeCommit, and CodeDeploy resources. Because these resources are specific to each service, they appear only with the matching service. If you include a resource that does not match any services in the `Action` element, then the resource appears in all action summaries.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1487980617000",
            "Effect": "Allow",
            "Action": [
                "codebuild:*",
                "codecommit:*",
                "codedeploy:*"
            ],
            "Resource": [
                "arn:aws:codebuild:us-east-2:123456789012:project/my-demo-
project",
                "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo",
                "arn:aws:codedeploy:us-
east-2:123456789012:application:WordPress_App",
                "arn:aws:codedeploy:us-east-2:123456789012:instance/AssetTag*"
```

```
            ]
        }
    ]
}
```

CodeBuild_CodeCommit_CodeDeploy *Policy Summary:*

Allow (3 of 370 services) Show remaining 367 services

Service	Access level	Resource	Request condition
CodeBuild	Full: Permissions management Limited: List, Read, Write	region| string like |us-east-2	None
CodeCommit	Full: Tagging Limited: List, Read, Write	ResourceSpecifier| string like |MyDemoRepo, region| string like |us-east-2	None
CodeDeploy	Full: Tagging Limited: List, Read, Write	Multiple	None

CodeBuild_CodeCommit_CodeDeploy CodeBuild (Allow) *Service Summary:*

< Services Actions in CodeBuild (24 of 53) Show remaining 29 actions

Read (4 of 9)

Action	Resource	Request condition
BatchGetBuildBatches	region| string like |us-east-2	None
BatchGetBuilds	region| string like |us-east-2	None
BatchGetProjects	region| string like |us-east-2	None
GetResourcePolicy	region| string like |us-east-2	None

Write (16 of 28)

Action	Resource	Request condition
BatchDeleteBuilds	region| string like |us-east-2	None
CreateProject	region| string like |us-east-2	None
CreateWebhook	region| string like |us-east-2	None
DeleteBuildBatch	region| string like |us-east-2	None
DeleteProject	region| string like |us-east-2	None
DeleteWebhook	region| string like |us-east-2	None
InvalidateProjectCache	region| string like |us-east-2	None
RetryBuild	region| string like |us-east-2	None
RetryBuildBatch	region| string like |us-east-2	None
StartBuild	region| string like |us-east-2	None
StartBuildBatch	region| string like |us-east-2	None
StopBuild	region| string like |us-east-2	None
StopBuildBatch	region| string like |us-east-2	None
UpdateProject	region| string like |us-east-2	None
UpdateProjectVisibility	region| string like |us-east-2	None
UpdateWebhook	region| string like |us-east-2	None

List (2 of 14)

CodeBuild_CodeCommit_CodeDeploy StartBuild (Write) *Action Summary:*

< Actions StartBuild action in CodeBuild

Resource	Region	Account	Request condition
region| string like |us-east-2	us-east-2	123456789012	None

Permissions required to access IAM resources

Resources are objects within a service. IAM resources include groups, users, roles, and policies. If you are signed in with AWS account root user credentials, you have no restrictions on administering IAM credentials or IAM resources. However, IAM users must explicitly be given permissions to administer credentials or IAM resources. You can do this by attaching an identity-based policy to the user.

> **Note**
>
> Throughout the AWS documentation, when we refer to an IAM policy without mentioning any of the specific categories, we mean an identity-based, customer managed policy. For details about policy categories, see the section called “Policies and permissions”.

Permissions for administering IAM identities

The permissions that are required to administer IAM groups, users, roles, and credentials usually correspond to the API actions for the task. For example, in order to create IAM users, you must have the `iam:CreateUser` permission that has the corresponding API command: `CreateUser`. To allow an IAM user to create other IAM users, you could attach an IAM policy like the following one to that user:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:CreateUser",
    "Resource": "*"
  }
}
```

In a policy, the value of the `Resource` element depends on the action and what resources the action can affect. In the preceding example, the policy allows a user to create any user (* is a wildcard that matches all strings). In contrast, a policy that allows users to change only their own access keys (API actions `CreateAccessKey` and `UpdateAccessKey`) typically has a `Resource`

element. In this case the ARN includes a variable (${aws:username}) that resolves to the current user's name, as in the following example:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListUsersForConsole",
            "Effect": "Allow",
            "Action": "iam:ListUsers",
            "Resource": "arn:aws:iam::*:*"
        },
        {
            "Sid": "ViewAndUpdateAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:UpdateAccessKey",
                "iam:CreateAccessKey",
                "iam:ListAccessKeys"
            ],
            "Resource": "arn:aws:iam::*:user/${aws:username}"
        }
    ]
}
```

In the previous example, ${aws:username} is a variable that resolves to the user name of the current user. For more information about policy variables, see IAM policy elements: Variables and tags.

Using a wildcard character (*) in the action name often makes it easier to grant permissions for all the actions related to a specific task. For example, to allow users to perform any IAM action, you can use iam:* for the action. To allow users to perform any action related just to access keys, you can use iam:*AccessKey* in the Action element of a policy statement. This gives the user permission to perform the CreateAccessKey, DeleteAccessKey, GetAccessKeyLastUsed, ListAccessKeys, and UpdateAccessKey actions. (If an action is added to IAM in the future that has "AccessKey" in the name, using iam:*AccessKey* for the Action element will also give the user permission to that new action.) The following example shows a policy that allows users

to perform all actions pertaining to their own access keys (replace *account-id* with your AWS account ID):

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "iam:*AccessKey*",
        "Resource": "arn:aws:iam::111122223333:user/${aws:username}"
    }
}
```

Some tasks, such as deleting a group, involve multiple actions: You must first remove users from the group, then detach or delete the group's policies, and then actually delete the group. If you want a user to be able to delete a group, you must be sure to give the user permissions to perform all of the related actions.

Permissions for working in the AWS Management Console

The preceding examples show policies that allow a user to perform the actions with the AWS CLI or the AWS SDKs.

As users work with the console, the console issues requests to IAM to list groups, users, roles, and policies, and to get the policies associated with a group, user, or role. The console also issues requests to get AWS account information and information about the principal. The principal is the user making requests in the console.

In general, to perform an action, you must have only the matching action included in a policy. To create a user, you need permission to call the `CreateUser` action. Often, when you use the console to perform an action, you must have permissions to display, list, get, or otherwise view resources in the console. This is necessary so that you can navigate through the console to make the specified action. For example, if user Jorge wants to use the console to change his own access keys, he goes to the IAM console and chooses **Users**. This action causes the console to make a `ListUsers` request. If Jorge doesn't have permission for the `iam:ListUsers` action, the console is denied access when it tries to list users. As a result, Jorge can't get to his own name and to his

own access keys, even if he has permissions for the `CreateAccessKey` and `UpdateAccessKey` actions.

If you want to give users permissions to administer groups, users, roles, policies, and credentials with the AWS Management Console, you need to include permissions for the actions that the console performs. For some examples of policies that you can use to grant a user for these permissions, see Example policies for administering IAM resources.

Grant permissions across AWS accounts

You can directly grant IAM users in your own account access to your resources. If users from another account need access to your resources, you can create an IAM role, which is an entity that includes permissions but that isn't associated with a specific user. Users from other accounts can then use the role and access resources according to the permissions you've assigned to the role. For more information, see Access for an IAM user in another AWS account that you own.

> **Note**
>
> Some services support resource-based policies as described in Identity-based policies and resource-based policies (such as Amazon S3, Amazon SNS, and Amazon SQS). For those services, an alternative to using roles is to attach a policy to the resource (bucket, topic, or queue) that you want to share. The resource-based policy can specify the AWS account that has permissions to access the resource.

Permissions for one service to access another

Many AWS services access other AWS services. For example, several AWS services—including Amazon EMR, Elastic Load Balancing, and Amazon EC2 Auto Scaling—manage Amazon EC2 instances. Other AWS services make use of Amazon S3 buckets, Amazon SNS topics, Amazon SQS queues, and so on.

The scenario for managing permissions in these cases varies by service. Here are some examples of how permissions are handled for different services:

- In Amazon EC2 Auto Scaling, users must have permission to use Auto Scaling, but don't need to be explicitly granted permission to manage Amazon EC2 instances.

- In AWS Data Pipeline, an IAM role determines what a pipeline can do; users need permission to assume the role. (For details, see Granting Permissions to Pipelines with IAM in the *AWS Data Pipeline Developer Guide*.)

For details about how to configure permissions properly so that an AWS service is able to accomplish the tasks you intend, refer to the documentation for the service you are calling. To learn how to create a role for a service, see Create a role to delegate permissions to an AWS service.

Configuring a service with an IAM role to work on your behalf

When you want to configure an AWS service to work on your behalf, you typically provide the ARN for an IAM role that defines what the service is allowed to do. AWS checks to ensure that you have permissions to pass a role to a service. For more information, see Grant a user permissions to pass a role to an AWS service.

Required actions

Actions are the things that you can do to a resource, such as viewing, creating, editing, and deleting that resource. Actions are defined by each AWS service.

To allow someone to perform an action, you must include the necessary actions in a policy that applies to the calling identity or the affected resource. In general, to provide the permission required to perform an action, you must include that action in your policy. For example, to create a user, you need add the CreateUser action to your policy.

In some cases, an action might require that you include additional related actions in your policy. For example, to provide permission for someone to create a directory in AWS Directory Service using the `ds:CreateDirectory` operation, you must include the following actions in their policy:

- `ds:CreateDirectory`
- `ec2:DescribeSubnets`
- `ec2:DescribeVpcs`
- `ec2:CreateSecurityGroup`
- `ec2:CreateNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:AuthorizeSecurityGroupIngress`

- `ec2:AuthorizeSecurityGroupEgress`

When you create or edit a policy using the visual editor, you receive warnings and prompts to help you choose all of the required actions for your policy.

For more information about the permissions required to create a directory in AWS Directory Service, see Example 2: Allow a User to Create a Directory.

Example policies for administering IAM resources

Following are examples of IAM policies that allow users to perform tasks associated with managing IAM users, groups, and credentials. This includes policies that permit users manage their own passwords, access keys, and multi-factor authentication (MFA) devices.

For examples of policies that let users perform tasks with other AWS services, like Amazon S3, Amazon EC2, and DynamoDB, see Example IAM identity-based policies.

Topics

- Allow a user to list the account's groups, users, policies, and more for reporting purposes
- Allow a user to manage a group's membership
- Allow a user to manage IAM users
- Allow users to set account password policy
- Allow users to generate and retrieve IAM credential reports
- Allow all IAM actions (admin access)

Allow a user to list the account's groups, users, policies, and more for reporting purposes

The following policy allows the user to call any IAM action that starts with the string `Get` or `List`, and to generate reports. To view the example policy, see IAM: Allows read-only access to the IAM console.

Allow a user to manage a group's membership

The following policy allows the user to update the membership of the group called *MarketingGroup*. To view the example policy, see IAM: Allows managing a group's membership programmatically and in the console.

Allow a user to manage IAM users

The following policy allows a user to perform all the tasks associated with managing IAM users but not to perform actions on other entities, such as creating groups or policies. Allowed actions include these:

- Creating the user (the `CreateUser` action).
- Deleting the user. This task requires permissions to perform all of the following actions: `DeleteSigningCertificate`, `DeleteLoginProfile`, `RemoveUserFromGroup`, and `DeleteUser`.
- Listing users in the account and in groups (the `GetUser`, `ListUsers` and `ListGroupsForUser` actions).
- Listing and removing policies for the user (the `ListUserPolicies`, `ListAttachedUserPolicies`, `DetachUserPolicy`, `DeleteUserPolicy` actions)
- Renaming or changing the path for the user (the `UpdateUser` action). The `Resource` element must include an ARN that covers both the source path and the target path. For more information on paths, see Friendly names and paths.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUsersToPerformUserActions",
            "Effect": "Allow",
            "Action": [
                "iam:ListPolicies",
                "iam:GetPolicy",
                "iam:UpdateUser",
                "iam:AttachUserPolicy",
                "iam:ListEntitiesForPolicy",
                "iam:DeleteUserPolicy",
                "iam:DeleteUser",
                "iam:ListUserPolicies",
                "iam:CreateUser",
                "iam:RemoveUserFromGroup",
                "iam:AddUserToGroup",
```

```
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:PutUserPolicy",
                "iam:ListAttachedUserPolicies",
                "iam:ListUsers",
                "iam:GetUser",
                "iam:DetachUserPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowUsersToSeeStatsOnIAMConsoleDashboard",
            "Effect": "Allow",
            "Action": [
                "iam:GetAccount*",
                "iam:ListAccount*"
            ],
            "Resource": "*"
        }
    ]
}
```

A number of the permissions included in the preceding policy allow the user to perform tasks in the AWS Management Console. Users who perform user-related tasks from the AWS CLI, the AWS SDKs, or the IAM HTTP query API only might not need certain permissions. For example, if users already know the ARN of policies to detach from a user, they do not need the `iam:ListAttachedUserPolicies` permission. The exact list of permissions that a user requires depends on the tasks that the user must perform while managing other users.

The following permissions in the policy allow access to user tasks via the AWS Management Console:

- `iam:GetAccount*`
- `iam:ListAccount*`

Allow users to set account password policy

You might give some users permissions to get and update the password policy of your AWS account. To view the example policy, see IAM: Allows setting the account password requirements programmatically and in the console.

Allow users to generate and retrieve IAM credential reports

You can give users permission to generate and download a report that lists all users in your AWS account. The report also lists the status of various user credentials, including passwords, access keys, MFA devices, and signing certificates. For more information about credential reports, see Generate credential reports for your AWS account. To view the example policy, see IAM: Generate and retrieve IAM credential reports.

Allow all IAM actions (admin access)

You might give some users administrative permissions to perform all actions in IAM, including managing passwords, access keys, MFA devices, and user certificates. The following example policy grants these permissions.

> **Warning**
>
> When you give a user full access to IAM, there is no limit to the permissions that user can grant to him/herself or others. The user can create new IAM entities (users or roles) and grant those entities full access to all resources in your AWS account. When you give a user full access to IAM, you are effectively giving them full access to all resources in your AWS account. This includes access to delete all resources. You should grant these permissions to only trusted administrators, and you should enforce multi-factor authentication (MFA) for these administrators.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "iam:*",
    "Resource": "*"
  }
}
```

IAM temporary delegation

Overview

Temporary delegation accelerates onboarding and simplifies management for products from Amazon and AWS Partners that integrate with your AWS accounts. Instead of manually configuring multiple AWS services, you can delegate temporary, limited permissions that allow the product provider to complete setup tasks on your behalf in minutes through automated deployment workflows. You maintain administrative control with approval requirements and permission boundaries, while product provider permissions automatically expire after the approved duration with no manual cleanup required. If the product requires persistent access for ongoing operations, the provider can use temporary delegation to create an IAM role with a permission boundary that defines the role's maximum permissions. All product provider activity is tracked through AWS CloudTrail for compliance and security monitoring.

To work with temporary delegation requests, you can initiate a request and review requests. For more information about monitoring, see notifications and CloudTrail logging.

> **Note**
>
> Temporary delegation requests can only be created by Amazon products and qualified AWS Partners that have completed the feature onboarding process. Customers review and approve these requests but cannot create them directly. If you are an AWS Partner looking to integrate IAM temporary delegation into your product, see the Partner Integration Guide for onboarding and integration instructions.

How temporary delegation works

Temporary delegation enables Amazon and AWS Partners to request temporary, limited access to your account. Upon your approval, they can use delegated permissions to take actions on your behalf. Delegation requests define specific permissions to AWS services and actions that the product provider needs to deploy or configure resources in your AWS account. These permissions are only available for a limited time and automatically expire after the duration specified in the request.

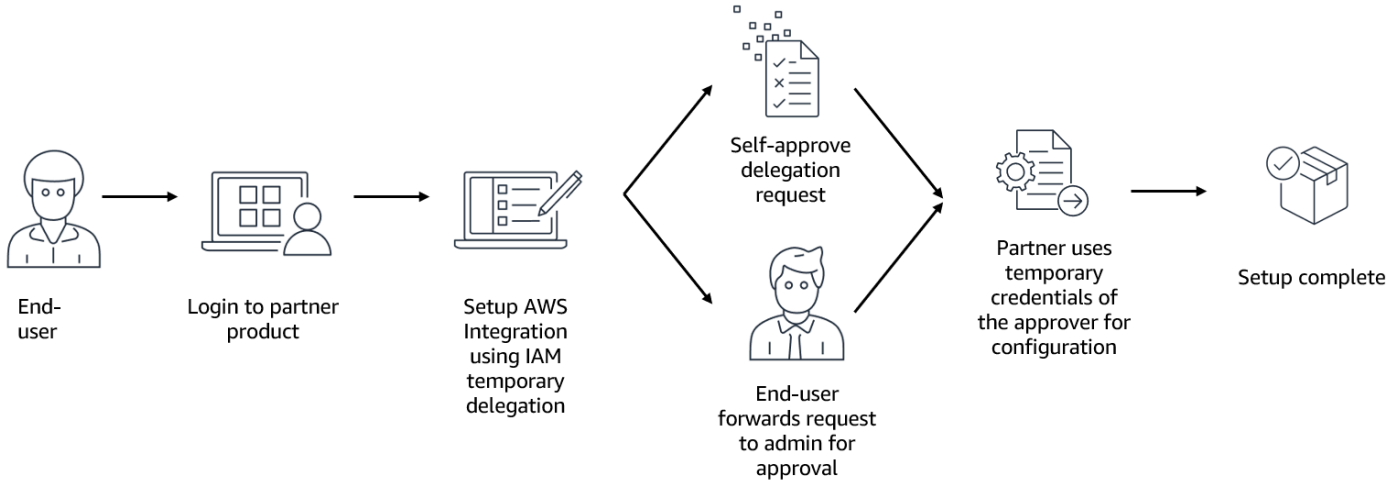
Note

The maximum duration for delegated access is 12 hours. However, root users can only approve delegation requests with a duration of 4 hours or less. If a request specifies more than 4 hours, you must use a non-root identity to approve the request. For details, see Permission simulation beta capability.

For ongoing tasks, such as reading from an Amazon S3 bucket, delegation requests can include the creation of an IAM role that allows continued access to resources and actions after the temporary access expires. Product providers must attach a permission boundary to any IAM role created through temporary delegation. Permission boundaries limit a role's maximum permissions but do not grant permissions on their own. You can review the permission boundary as part of the request before approving it. For details, see Permissions boundaries.

The process works as follows:

1. You log in to an Amazon or AWS Partner product to integrate it with your AWS environment.
2. The product provider initiates a delegation request on your behalf and redirects you to the AWS Management Console.
3. You review the requested permissions and determine whether to approve, deny, or forward the request to your administrator.
4. After you or your administrator approves the request, the product provider can obtain approver's temporary credentials to perform the required tasks.
5. Product provider access automatically expires after the specified time period. However, any IAM role created through the temporary delegation request persists beyond this period, allowing the product provider to continue accessing resources and actions for ongoing management tasks.



> **Note**
>
> You can only delegate permissions to a product provider if you have permissions to the services and actions included in the temporary delegation request. If you don't have access to the requested services and actions, the product provider does not receive these permissions when you approve the request.

If the permission check shows it is likely to succeed, you can approve the temporary delegation request and continue with the workflow.

If the permission check shows that you might not have sufficient permissions, forward the request to your administrator for approval. We recommend notifying your administrator about this request using your preferred method such as an email or a ticket.

After your administrator approves the request, what happens next depends on the product provider's configuration:

- If the product provider requested immediate access, they automatically receive temporary permissions and the access duration begins.
- If the product provider requested release by the owner (initial recipient), you must return to the request to explicitly share temporary account access before the access duration begins. Product providers typically use this option when they need additional input from you, such as resource selection or configuration details, to complete the required task.

Initiate a temporary delegation request

You can initiate a temporary delegation request only from supported Amazon or AWS Partner products. During a workflow that supports temporary delegation, you will be prompted to grant the product provider temporary, limited permissions to configure required AWS resources in your account. This automated approach provides a more streamlined experience by eliminating the need for you to configure these resources manually.

You can review delegation request details such as the specific IAM roles, policies, and AWS services the product provider needs before granting access. You can either approve the request yourself if you have sufficient permissions, or forward the request to your account administrator for approval. All product provider access is time-bounded and can be monitored and revoked as needed.

To initiate a temporary delegation request

1. Navigate to the console of a supported product from Amazon or AWS Partners that requires integration with your AWS account.
2. Select *Deploy with IAM temporary delegation*. Note that the option name might vary across supported products. Refer to the documentation of the product provider for details.

   > **Note**
   >
   > If you are not already signed in to the AWS Management Console, a new window opens to the AWS sign-in page. We recommend that you sign in to your AWS account before initiating the temporary delegation request from the product console. For more information about how to sign in based on your user type and the AWS resources that you want to access, see the AWS Sign-In User Guide.

3. Review the request details to confirm the product provider's product name and AWS account. You can also review the AWS identity that the product provider will use to perform actions on your behalf.
4. Review the *Access details* for the permissions that will be temporarily delegated by approving this request.

   - The *Permissions summary* section provides a high-level overview generated by AI that can help you understand what categories of AWS services can be accessed and what types of actions can be performed in each service.

- Choose *View JSON* to review specific permissions the product provider needs to deploy in your AWS account, including access scope and resource limitations.
- If the product provider creates an IAM role as part of a temporary delegation request, a permission boundary must be attached to the role. These IAM roles have permissions that continue to allow access to resources and actions after the requested access duration expires. Choose *View details* to review the permission boundary, which defines the maximum permissions that the role can have. The product provider will apply additional policies to the role during creation that define its actual permissions. These policies might appear more narrowly focused, or broader, than the boundary depending on how the product provider defines them. However, the permission boundary guarantees that the role's effective permissions will never exceed what you see during request approval, regardless of what policies are attached to the role. For more information, see Permissions boundaries.

5. Review the permission simulation results. The permission simulation capability automatically evaluates your identity's permissions against those included in the request. Based on this analysis, a recommendation is displayed indicating whether to approve the request with your current identity or forward the request to an administrator. For details, see Permission simulation beta capability .
6. In the dialog, select how you would like to proceed.

   - Select *Allow access* when your identity has sufficient permissions to allow the product provider to perform onboarding procedures on your behalf. When you select this option, the product provider's access duration starts after you provide access.
   - Select *Request approval* if your identity doesn't have sufficient permissions to allow the product provider to perform onboarding procedures on your behalf. Then, choose *Create approval request*. When you select this option, a temporary delegation request link is created that you can share with your account administrator. Your administrator can access the AWS Management Console or use the access link to Review Temporary delegation requests to approve the request and share temporary access with the requestor.

> **Note**
>
> Granting product provider access requires two actions: accepting the delegation request (`AcceptDelegationRequest`) and releasing the exchange token (`SendDelegationToken`). The AWS Management Console performs both steps

automatically when you approve a request. If you use the AWS CLI or API, you must execute both steps separately.

Permission simulation capability -beta

When you receive a temporary delegation request, you can either approve it yourself or forward it to your account administrator for approval. You can only delegate permissions to a product provider if you have permissions to the services and actions included in the temporary delegation request. If you don't have access to the requested services and actions, the product provider won't receive those permissions even if they're included in the request.

For example, a temporary delegation request requires the ability to create an Amazon S3 bucket, start and stop instances in Amazon EC2, and assume an IAM role. The identity that approves the request can start and stop instances in Amazon EC2, and assume an IAM role, but does not have permission to create an Amazon S3 bucket. When this identity approves the request, the product provider is not able to create an Amazon S3 bucket even though these permissions were included in the temporary delegation request.

Since you can only delegate permissions that you already possess, it's crucial to evaluate whether you have the requested permissions before approving. The permission simulation beta capability helps with this evaluation by comparing your permissions against those included in the request. The assessment indicates whether you can approve the request with your current identity or need to forward it to an administrator. If the analysis cannot validate that you have sufficient permissions, forward the request to an administrator for review. This assessment is based on simulated permission analysis and might differ from your live AWS environment, so review requested permissions carefully before proceeding.

Next steps

After you initiate a temporary delegation request, you can manage and monitor the request through its lifecycle. The following procedures help you track, approve, and control temporary access:

- Review Temporary delegation requests – Monitor the status of your access requests, and view detailed information about requests to approve or deny temporary delegation requests.
- Revoke temporary delegation access – Immediately terminate active temporary delegation sessions before they expire naturally.

Review Temporary delegation requests

After initiating a temporary delegation request, you can monitor, approve, and reject requests in the IAM console. The Temporary delegation requests page provides a centralized view of all requests, including those that are pending approval, completed, or rejected. As an administrator, you can review these requests to grant product providers access to AWS resources or reject them based on your organization's security policies, business requirements, or compliance standards. This visibility helps you track the lifecycle of product provider access and maintain oversight of temporary permissions.

> **Note**
>
> You must have the iam:AcceptDelegationRequest permission to approve temporary delegation requests.

To approve a temporary delegation request

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose *Temporary delegation requests*.
3. The main page displays a list of your temporary delegation requests with the following information:
   - *Request ID* – Unique identifier for the request
   - *Status* – Current status (Pending, Approved, Rejected, Shared, Expired)
   - *Requestor* - Product provider associated with the request
   - *Initiated by* - IAM principal in the account who initiated the request for the product provider
   - *Request created*– When the request was submitted
   - *Request expires* – When the request expired or will expire
4. (Optional) Use the filter options to view requests by status:
   - *All requests* – View all your requests regardless of status
   - *Pending* – View requests awaiting administrator approval
   - *Approved* – View approved requests
   - *Shared* – View requests for which access has been shared

- *Rejected* – View rejected requests with rejection reasons

5. To view detailed information about a specific request or to review a pending request for approval, choose the request ID.
6. Review the detailed request information:
   - Product provider information
   - Request reason and justification
   - Requested duration
   - Requested AWS permissions
7. If you are an administrator reviewing a pending request, choose one of the following options:
   - To approve the request, choose *Approve*. In the approval dialog, you can view the results of the permission simulation. For more information, see Permission simulation beta capability. After confirming access duration and your AWS identity, choose *Approve* to grant access. If the product provider requested immediate access, they automatically receive temporary permissions and the access duration begins. Otherwise, notify the person who initiated the request to release access to the product provider.
   - To reject the request, choose *Reject*.
     - In the rejection dialog, provide a clear reason for the rejection to help the requester understand why their request was denied.
     - Choose *Reject* to deny access.
8. The request list automatically refreshes to show the most current status information. You can also manually refresh the page to check for status updates.

Revoke Temporary delegation access

Although product provider access sessions are designed to expire automatically after their approved duration, you might need to terminate access immediately in certain situations. Revoking active product provider access provides an emergency control mechanism when security concerns arise, when the product provider's work is completed early, or when business requirements change. Both request initiators and administrators can revoke access to maintain security and operational control.

To revoke temporary delegation access

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose *Temporary delegation requests*.
3. Locate the request ID for the access session you want to revoke.
4. Choose *Actions* and then choose *Revoke access*.
5. In the dialog, choose *Revoke access* to confirm that you want to immediately terminate the access session.

After revoking access, the product provider will no longer be able to access your AWS resources. The revocation is logged in AWS CloudTrail for audit purposes.

> **Important**
>
> Revoking access immediately terminates the product provider access session. Any ongoing work or processes using the access will be interrupted. Make sure that revocation won't disrupt critical operations.

> **Note**
>
> You cannot revoke access for requests that were approved using a root user. AWS recommends that you avoid using a root user to approve delegation requests. Use an IAM role with appropriate permissions instead.

Managing Permissions for Delegation Requests

Administrators can grant IAM principals permissions to manage delegation requests from product providers. This is useful when you want to delegate approval authority to specific users or teams in your organization, or when you need to control who can perform specific actions on delegation requests.

The following IAM permissions are available for managing delegation requests:

Permission	Description
iam:AssociateDelegationRequest	Associate an unassigned delegation request with your AWS account
iam:GetDelegationRequest	View details of a delegation request
iam:UpdateDelegationRequest	Forward a delegation request to an administrator for approval
iam:AcceptDelegationRequest	Approve a delegation request
iam:SendDelegationToken	Release the exchange token to the product provider after approval
iam:RejectDelegationRequest	Reject a delegation request
iam:ListDelegationRequests	List delegation requests for your account

Note

By default, IAM principals who initiate a delegation request are automatically granted permissions to manage that specific request. They can associate it with their account, view request details, reject a request, forward it to an administrator for approval, release the exchange token to the product provider after admin approval, and list delegation requests that they own.

Notifications

IAM temporary delegation integrates with AWS User Notifications to help you stay informed about delegation request state changes. Notifications are particularly useful for administrators who need to review and approve delegation requests.

With AWS User Notifications, you can configure alerts to be delivered through multiple channels, including email, Amazon Simple Notification Service (SNS), AWS Chatbot for Slack or Microsoft Teams, and AWS Console Mobile Application. This ensures that the right people are notified at the right time, enabling faster response to pending approvals or awareness of access changes. You can also customize which events trigger notifications based on your organization's needs and security requirements.

Available Notification Events

You can subscribe to receive notifications for the following IAM temporary delegation events:

- IAM Temporary Delegation Request Created
- IAM Temporary Delegation Request Assigned
- IAM Temporary Delegation Request Pending Approval
- IAM Temporary Delegation Request Rejected
- IAM Temporary Delegation Request Accepted
- IAM Temporary Delegation Request Finalized
- IAM Temporary Delegation Request Expired

Configuring Notifications

To configure notifications for IAM temporary delegation events:

1. Open the AWS User Notifications console
2. Create or update a notification configuration
3. Select AWS IAM as the service
4. Choose which delegation request events you want to be notified about
5. Configure your delivery channels (email, AWS Chatbot, etc.)

For detailed instructions on configuring AWS User Notifications, including setting up delivery channels and managing notification rules, see the AWS User Notifications documentation.

CloudTrail

All actions performed by product providers using temporary delegated access are automatically logged in AWS CloudTrail. This provides complete visibility and auditability of product provider

activity in your AWS account. You can identify which actions were taken by product providers, when they occurred, and which product provider account performed them.

To help you distinguish between actions taken by your own IAM principals and those taken by product providers with delegated access, CloudTrail events include a new field called `invokedByDelegate` under the `userIdentity` element. This field contains the AWS account ID of the product provider, making it easy to filter and audit all delegated actions.

CloudTrail Event Structure

The following example shows a CloudTrail event for an action performed by a product provider using temporary delegated access:

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:Role-Session-Name",
        "arn": "arn:aws:sts::111122223333:assumed-role/Role-Name/Role-Session-Name",
        "accountId": "111122223333",
        "accessKeyId": "[REDACTED:AWS_ACCESS_KEY]",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2024-09-09T17:50:16Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedByDelegate": {
            "accountId": "444455556666"
        }
    },
    "eventTime": "2024-09-09T17:51:44Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "GetUserPolicy",
    "awsRegion": "us-east-1",
    "requestParameters": {
```

```
        "userName": "ExampleIAMUserName",
        "policyName": "ExamplePolicyName"
    },
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

The `invokedByDelegate` field contains the AWS account ID of the product provider who performed the action using delegated access. In this example, account 444455556666 (the product provider) performed an action in account 111122223333 (the customer account).

IAM temporary delegation for AWS Partners

Overview

IAM temporary delegation enables AWS customers to seamlessly onboard and/or integrate AWS Partner products into their AWS environment through interactive, guided workflows. Customers can grant AWS Partners limited, temporary access to configure required AWS services, reducing onboarding friction and accelerating time to value.

To integrate temporary delegation into your product, learn about understanding your integration, review understanding permissions, and follow the policy evaluation guidelines. For sample IAM policies, see policy templates. When you are ready to implement, see building your integration.

IAM temporary delegation enables partners to:

- Streamline customer onboarding with automated resource provisioning
- Reduce integration complexity by eliminating manual configuration steps
- Build trust through transparent, customer-approved permissions
- Enable ongoing operations with long-term access patterns using permission boundaries

How it works

1. *Partner creates a delegation request* - Partners create a request specifying what permissions they need and for how long
2. *Customer reviews in AWS Console* - Customer sees exactly what permissions partner is requesting and why

3. *Customer approves* - Customer approves the request and releases an exchange token. The token is sent to partner on this specified SNS topic.
4. *Partner receives temporary credentials* - Partners exchange the token for temporary AWS credentials
5. *Partner configures resources* - Partners use the credentials to set up required resources in the customer's account

Partner qualification

To qualify for temporary delegation integration, a partner must meet the following requirements:

- *ISV Accelerate participation* – You must be enrolled in the ISV Accelerate (ISVA) program.
- *AWS Marketplace listing* – Your product must be listed in the AWS Marketplace with a "Deployed on AWS" badge.

Onboarding process

Complete the following steps to integrate temporary delegation into your product:

1. *Step 1: Review requirements*

   Review this documentation to understand the qualification requirements and complete the partner questionnaire below.
2. *Step 2: Submit your onboarding request*

   Send an email to aws-iam-partner-onboarding@amazon.com or contact your AWS representative. Include your completed partner questionnaire with all required fields from the table below.
3. *Step 3: AWS validation and review*

   AWS will:

   - Validate that you meet the qualification criteria
   - Review your policy templates and permission boundaries
   - Provide feedback on your submitted artifacts
4. *Step 4: Refine your policies*

Respond to AWS feedback and submit updated policy templates or permission boundaries as needed.

5. *Step 5: Complete registration*

   After approval, AWS will:

   - Enable API access for your specified accounts
   - Share ARNs for your policy template and permissions boundary (if applicable)

   You will receive confirmation when onboarding is complete. You can then access temporary delegation APIs, CreateDelegationRequest and GetDelegatedAccessToken from your registered accounts and begin integrating delegation request workflows into your product.

Partner questionnaire

The following table lists the information required for partner onboarding:

Information	Description	Required
Partner Central AccountID	Account ID of your registered AWS account on AWS Partner Central.	Yes
PartnerId	Partner ID provided by AWS Partner Central.	No
AWS Marketplace Product Id	Product ID for your product provided by AWS Partner Central.	Yes
AWS accountIDs	The list of your AWS Account IDs that you want to use to call temporary delegation APIs. This should include both your production and non-production/test accounts.	Yes
Partner name	This name is displayed to customers in the AWS Management Console when they review your temporary delegation request.	Yes
Contact email(s)	One or more email addresses for contacting you about your integration.	Yes

Information	Description	Required
Requestor Domain	Your domain (for example, www.example.com)	Yes
Integration description	Brief description of the use case that you want to address using this feature. You can include reference links to your documentation or other public material.	Yes
Architecture diagram	Architecture diagram illustrating your integration use case(s).	No
Policy template	You must register at least one policy template for this feature. The policy template defines the temporary permissions you want to request in customers' AWS accounts. For more information, see Policy template section.	Yes
Policy template name	Name of the policy template you want to register.	Yes
Permissions Boundary	If you want to create IAM roles in customers' accounts using temporary permissions, you must register a permission boundary with IAM. Permission boundarie s will be attached to the IAM roles that you create to limit the maximum permissions on the role. You can use selected AWS managed policies as a permission boundary or register a new custom permission boundary (JSON). For more information, see Permissions Boundary section.	No
Permission Boundary Name	The name of your permission boundary. The format is: arn:aws:iam::partner:policy/permission_boundary/ <partner_domain>/<policy_name>_<date> The policy name must include the creation date as a suffix. The name cannot be updated after the permission boundary is created. If you are using an existing AWS managed policy, provide the managed policy ARN instead.	No

Information	Description	Required
Permissio n Boundary Description	Description for the permission boundary. This descripti on cannot be updated after the permission boundary is created.	No

Understanding your integration

After completing the onboarding process, you can build your integration with IAM temporary delegation. A complete integration typically involves three main categories of work:

1. User Experience and Workflow Design

Build a front-end experience in the partner application that guides customers through the temporary delegation workflow. Partner application should:

- Present a clear onboarding or configuration flow where customers can grant temporary access. Label this action clearly, such as "Deploy with IAM temporary delegation".
- Redirect customers to the AWS Management Console to review and approve the delegation request using the console link returned by the CreateDelegationRequest API
- Provide appropriate messaging about what permissions are being requested and why. Customers can see this message on the delegation request details page.
- Handle the customer's return to your application after they complete the approval in AWS.

Best Practices for Temporary Delegation Requests

When implementing temporary delegation for your partner application, follow the below defined best practices to help customers verify request authenticity and accuracy.

1. Include User-Identifiable Context in Request Messages

You should include user-specific context in delegation request message. This information helps users identify the requests from their existing partner workflow and distinguish legitimate requests. Suggested information in request message:

- Customer's account identifier or username in your service
- Workspace name, subscription ID, or organization identifier

- Specific resource being accessed (cluster name, project name, environment)
- Unique transaction or request identifier generated for this delegation attempt

Example request message:

```
Request from Partner A workspace "production-analytics"
Account: user@example.com
Workspace ID: 1234ABCD
Cluster: ml-training-cluster-01
Request ID: 1111-2222-3333-4444
```

2. Optional: Include AWS Account ID when Initiating Delegation

If the customer account ID is available, include it in the delegation request. This verification step creates an additional binding between the customer's intent and the delegation token.

3. Design Request Messages for Security Verification

Structure your request messages to enable customers to confidently verify legitimacy before granting access. User experience requirements:

- Display the request message prominently in your application interface before redirecting to AWS
- Use clear, descriptive language that connects directly to the customer's current action
- Avoid generic messages that could apply to any delegation request
- Include sufficient detail for customers to verify the request matches their intended workflow
- Display the AWS account ID that will receive delegated access
- Provide a clear explanation of what permissions your application will request

Include a confirmation notice in your application before redirecting customers to the AWS consent screen:

```
## Redirecting to IAM Temporary Delegation
You are about to grant [Your Service Name] temporary access to your AWS account.

Before clicking "Allow" on the AWS consent screen:
• Verify the request details match your current action
```

```
- Confirm the AWS account ID matches your intended account
- Make sure you initiated this request from [Your Service Name]
```

4. Session Binding Recommendations

While AWS IAM handles the core authorization flow, implement these practices in your application to strengthen session integrity:

- Generate unique, single-use request identifiers for each delegation attempt
- Associate delegation requests with the customer's active application session
- Implement appropriate request expiration depending on your delegation process
- Validate that delegation callbacks match the originating request context
- Log all delegation request initiations and completions for security monitoring

By following these best practices, you help create a more secure temporary delegation experience.

2. API Integration

Use IAM temporary delegation APIs to send and manage delegation requests. After your AWS accounts are registered, you can access the following APIs:

- *IAM CreateDelegationRequest* – Creates a delegation request for a customer's AWS account. This API returns a console link that you redirect customers to for reviewing and approving the request.
- *AWS STS GetDelegatedAccessToken* – Retrieves temporary AWS credentials after a customer approves your delegation request. Use these credentials to perform actions in the customer's account.

Your integration should handle the complete lifecycle of delegation requests, including creating requests, monitoring their status, and retrieving temporary credentials when approved.

3. Resource Configuration and Orchestration

After you obtain temporary credentials, orchestrate the necessary workflows to configure resources in the customer's AWS account. This might include:

- Calling AWS service APIs directly to create and configure resources

- Deploying infrastructure using AWS CloudFormation templates
- Creating IAM roles for ongoing access (requires using permission boundaries). As a defense-in-depth measure, we recommend that you use dynamic role names, for example, a name with a randomly generated suffix. This approach makes the role name in each customer account unique and unpredictable.

Your orchestration logic should be idempotent and handle failures gracefully, as customers might need to retry or modify their delegation approvals.

Understanding permissions

As part of the feature onboarding process, you will need to register policies with IAM that define the permissions you want to request in customers' AWS accounts. The registration process provides a more consistent experience for customers and helps avoid common pitfalls in policy authoring.

During registration, AWS evaluates your policies against a set of validations. These validations are intended to standardize policy formatting and structure, and provide basic protections against known anti-patterns. The validations also reduce the risk of privilege escalations, unintended cross-account access, and broad access to high-value resources in customer accounts.

Permission Types

AWS will consider two categories of permissions: temporary and long-term.

Temporary Permissions

Temporary permissions limit the permissions assigned to any temporary delegated access sessions. Temporary permissions are described in policy templates that are applied to the delegated session. The templates support parameters that you supply when you create a delegation request. These parameter values are then bound to the session. The temporary permissions work in the same way as the session policies available in AWS STS today: the policies limit the capability of the underlying user, but do not grant any additional access. For more information, see the AWS STS documentation on session policies.

Long-Term Permissions

Long-term permissions limit the permissions of any roles that are created or managed through temporary access. Long-term permissions are implemented as IAM permissions boundaries. You might submit one or more permission boundaries to AWS as part of onboarding. After approval, AWS will share a policy ARN with you that you can reference in your policies.

These boundary policies have two noteworthy features. First, they are immutable. If you want to update permissions, you can register a new permissions boundary. You can then attach the new permissions boundary to your customers' roles by sending a new delegation request. Second, the policies are not templated. Since the same boundary policy is globally shared, they cannot be altered on a per-customer basis.

> **Important**
>
> Permission boundaries have a maximum size limit of 6,144 characters.

> **Note**
>
> If you would like to update a permission boundary or policy template, contact IAM at aws-iam-partner-onboarding@amazon.com. After the new permission boundary is registered, you can then send a delegation request to customers to update the IAM role and attach the newly registered permission boundary. See the Examples section for more details.

Example Use Case: Data Processing Workload

Consider a product provider that runs a data processing workload in customer accounts. The provider needs to set up infrastructure during initial onboarding, but also requires ongoing access to operate the workload.

Temporary permissions (for initial setup):

- Create Amazon EC2 instances, VPC, and security groups
- Create an Amazon S3 bucket for processed data
- Create an IAM role for ongoing operations
- Attach a permission boundary to the IAM role

Long-term permissions (IAM role with permission boundary for ongoing operations):

- Start and stop Amazon EC2 instances to run processing jobs
- Read input data from Amazon S3 bucket
- Write processed results to Amazon S3 bucket

The temporary permissions are used once during onboarding to configure the infrastructure. The IAM role created during this process has a permission boundary that limits its maximum permissions to only the operations needed for ongoing workload management. This ensures that even if the role's policies are modified, it cannot exceed the permissions defined in the boundary.

Policy evaluation guidelines

AWS will evaluate your submitted policies against a set of guidelines. The same evaluation guidelines apply to both policy templates and permission boundaries, with minor differences noted where appropriate.

For the purposes of evaluation, services are separated into distinct groups. The most important distinction is for security-sensitive services, which manage access, credentials, and keys. Policies granting access to these services need to be focused narrowly on the work being done. Security-sensitive services include the following: AWS Identity and Access Management (IAM), AWS Key Management Service (KMS), AWS Resource Access Manager (RAM), AWS IAM Identity Center, AWS Organizations, and AWS Secrets Manager.

A secondary distinction is services that can access data across account boundaries. Policies for these services must include protections to prevent unintentional cross-account access.

Common validations

All policy statements must follow these guidelines:

- All statements must include Effect, Action (or NotAction), Resource, and Condition fields in that order
- All actions within a single statement must be listed alphabetically
- All ARNs included in the policy must follow the syntax defined in public documentation for the relevant services
- NotAction fields can only be used in Deny statements
- Actions in Allow statements must include a service code. Generic wildcards ("*") are not allowed

Security-sensitive service restrictions

The following restrictions apply to security-sensitive services mentioned above:

- Actions in Allow statements must be more specific than [service]:*

- Actions in Allow statements for temporary access policy templates must not contain wildcards
- Sensitive actions, such as iam:PassRole or iam:CreateServiceLinkedRole, require additional scoping, such as specific resources or conditional checks. These actions include:
  - IAM role passing
  - IAM role modification actions
  - IAM policy modification actions
  - AWS KMS write or cryptographic operations
  - AWS RAM write or share operations
  - AWS Secrets Manager operations for retrieving or modifying secrets, or modifying resource policies
- Other actions might use a wildcard resource, such as iam:ListUsers or iam:GetPolicy
- Actions that manage credentials, such as iam:CreateAccessKey, are blocked

IAM-specific restrictions

For IAM:

- Only limited write operations are allowed for IAM roles and policies. You can not request permissions on other IAM resources such as users, groups, and certificates.
- Policy attachment or inline policy management actions are limited to roles with a permission boundary. Permission boundaries must be partner-supplied or on a list of allowed AWS managed policies. AWS managed policies might be allowed if they do not grant highly privileged or administrative permissions. For example, AWS managed policies for specific job functions or the SecurityAudit policy might be acceptable. AWS will review each AWS managed policy on a case-by-case basis during the onboarding process.
- Policy management is only allowed for policies with a partner-specific path: arn:aws:iam::@{AccountId}:policy/partner_domain.com/[feature]*
- Tags might only be applied during resource creation, and only for roles and policies
- iam:PassRole checks must match a specific name or path prefix

AWS STS-specific restrictions

For AWS STS:

- sts:AssumeRole must be scoped to a specific role ARN, role ARN prefix, or limited to a set of accounts or organization ID/organizational unit

IAM Identity Center restrictions

For AWS IAM Identity Center, the following actions are blocked:

- All actions dealing with permissions management (e.g., sso:AttachCustomerManagedPolicyReferenceToPermissionSet)
- User, group, and membership modifications for AWS Identity Store
- Tag management

AWS Organizations restrictions

For AWS Organizations, only read actions will be allowed.

Additional service-specific validations

- Actions that acquire secrets or credentials, such as glue:GetConnection or redshift:GetClusterCredentials, must have conditions matching either full ARNs, ARN prefixes, or tags
- For Amazon Redshift: redshift:GetClusterCredentials is only allowed on a specific database name, and redshift:GetClusterCredentialsWithIAM is only allowed on a specific workgroup name

> **Note**
>
> When managing IAM resources in the account, we recommend using a path that indicates your name, e.g., arn:aws:iam::111122223333:role/partner.com/rolename. This will help differentiate the resources associated with your integration and make discovery, audit, and analysis easier for customers.

Cross-account access requirements

Statements that potentially allow cross-account access must include at least one of the following:

- A condition specifying the account or organization for the resource (e.g., aws:ResourceOrgId matching one or more expected values)
- A Resource field that includes a specific account (e.g., arn:aws:sqs:*:111122223333:*)
- A Resource field that includes a non-wildcard account and a full resource name (e.g., arn:aws:s3:::full-bucket-name)

> **Note**
>
> Cross-account access is a sensitive capability that requires clear business justification. AWS will carefully review the need for cross-account access during the onboarding process.

Policy templates

Policy templates are a new IAM construct designed for defining temporary permissions that partners request in customers' accounts. Like regular IAM policies, they define permissions using statements with Effect, Action, Resource, and Condition elements. The key difference is that policy templates include parameters (such as @{bucketName}) that are replaced with actual values when you create a delegation request.

How Policy Templates Work

As part of the onboarding process, you register your policy templates with AWS. AWS assigns each template a unique ARN that you reference when creating delegation requests.

When you create a delegation request, you specify:

- The policy template ARN
- Parameter values to substitute into the template

AWS combines the template with your parameter values to generate a standard IAM policy. Customers review this final rendered policy when approving your delegation request, seeing exactly what permissions will be granted.

> **Note**
>
> The final rendered policy has a maximum size limit of 2048 characters.

Here's a simple example showing how template substitution works.

Policy Template:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::@{bucketName}/*"
        }
    ]
}
```

Parameters Provided in Delegation Request:

```
{
    "Name": "bucketName",
    "Values": ["customer-data-bucket"],
    "Type": "String"
}
```

Final rendered policy (what customers see):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::customer-data-bucket/*"
        }
    ]
}
```

Template Syntax

Policy templates use two key features to provide flexibility: parameter substitution and conditional statements. Parameter substitution allows you to define placeholders in your template that are replaced with actual values when creating a delegation request. Conditional statements allow you to include or exclude entire policy statements based on parameter values.

Parameter Substitution and Types

Use the @{parameterName} syntax to define parameters in your policy template. When creating a delegation request, you must specify the type for each parameter.

String

A single value that is substituted directly into the template.

Template:

```
"Resource": "arn:aws:s3:::@{bucketName}/*"
```

Parameters:

```
{
    "Name": "bucketName",
    "Values": ["my-bucket"],
    "Type": "String"
}
```

Rendered result:

```
"Resource": "arn:aws:s3:::my-bucket/*"
```

StringList

Multiple values that generate multiple resource entries. When a StringList parameter is used in a resource ARN, it expands to create separate resource entries for each value.

Template:

```
"Resource": "arn:aws:s3:::@{bucketNames}/*"
```

Parameters:

```
{
    "Name": "bucketNames",
    "Values": ["bucket-1", "bucket-2"],
    "Type": "StringList"
}
```

Rendered result:

```
"Resource": [
    "arn:aws:s3:::bucket-1/*",
    "arn:aws:s3:::bucket-2/*"
]
```

Cross-Product Behavior

When multiple parameters are used in the same resource ARN, StringList parameters create a cross-product of all combinations.

Template:

```
"Resource": "arn:aws:s3:::@{bucketNames}/@{prefix}/*"
```

Parameters:

```
[
    {
        "Name": "bucketNames",
        "Values": ["bucket-1", "bucket-2"],
        "Type": "StringList"
    },
    {
        "Name": "prefix",
        "Values": ["data"],
        "Type": "String"
    }
]
```

Rendered result:

```
"Resource": [
    "arn:aws:s3:::bucket-1/data/*",
    "arn:aws:s3:::bucket-2/data/*"
]
```

Conditional Statements

Use the @Enabled directive to conditionally include or exclude entire statements based on parameter values.

Syntax:

- @Enabled: "parameterName" - Include the statement when parameter value is "True"
- @Enabled: "!parameterName" - Include the statement when parameter value is NOT "True" (negation)

Template:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:GetObject"],
            "Resource": "*"
        },
        {
            "@Enabled": "ENABLE_S3_WRITE",
            "Effect": "Allow",
            "Action": ["s3:PutObject"],
            "Resource": "arn:aws:s3:::@{bucketName}/*"
        }
    ]
}
```

Parameters (when ENABLE_S3_WRITE is "True"):

```
[
    {
        "Name": "bucketName",
```

```
        "Values": ["my-bucket"],
        "Type": "String"
    },
    {
        "Name": "ENABLE_S3_WRITE",
        "Values": ["True"],
        "Type": "String"
    }
]
```

Rendered result:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:GetObject"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["s3:PutObject"],
            "Resource": "arn:aws:s3:::my-bucket/*"
        }
    ]
}
```

Parameters (when ENABLE_S3_WRITE is "False"):

```
[
    {
        "Name": "bucketName",
        "Values": ["my-bucket"],
        "Type": "String"
    },
    {
        "Name": "ENABLE_S3_WRITE",
        "Values": ["False"],
        "Type": "String"
    }
]
```

Rendered result:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:GetObject"],
            "Resource": "*"
        }
    ]
}
```

When ENABLE_S3_WRITE is set to "True", the conditional statement is included. When set to "False", the statement is excluded from the rendered policy.

Additional Examples

The following examples demonstrate common patterns for using policy templates in temporary delegation. They focus on creating IAM roles with permission boundaries for long-term access and show different strategies for scoping permissions to specific resources. These examples illustrate how to balance flexibility with security by using techniques such as ARN prefixes, resource tagging, and permission boundary updates.

Example 1: Granting Long-Term Access to Specific Resources

The following permission boundary is submitted as "SQSAccessorBoundary" for "partner.com":

```
{
    "Effect": "Allow",
    "Action": [
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "sqs:SendMessage"
    ],
    "Resource": "arn:aws:sqs:*:*:*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
```

```
}
```

> **Note**
>
> This includes a same-account condition to avoid granting access to queues in other accounts with open resource policies. A direct reference to the customer's account ID cannot be included because the boundary is shared across all customers and cannot be templated.

Since this is the first version of this policy, its ARN is arn:aws:iam::partner:policy/permissions-boundary/partner.com/SQSAccessorBoundary_2025_01_15

The following policy template is submitted for temporary access permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ListQueues"
            ],
            "Resource": "arn:aws:sqs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:PutRolePermissionsBoundary",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::@{AccountId}:role/partner.com/SQSAccessor",
            "Condition": {
                "StringEquals": {
                    "iam:PermissionsBoundary": "arn:aws:iam::partner:policy/
permissions-boundary/partner.com/SQSAccessorBoundary_2025_01_15"
                }
            }
        }
    ]
```

```
}
```

Example 2: Using ARN Prefixes

The permission boundary can specify a resource ARN prefix to limit access:

```
"Resource": "arn:aws:sqs:*:@{AccountId}:PartnerPrefix*"
```

This limits access to only the resources with that prefix, reducing the scope of accessible resources.

Example 3: Using Tags for Resource Access Control

You can tag resources during temporary delegated access and rely on those tags for long-term access control.

Permission boundary allowing access to tagged resources:

```
{
    "Effect": "Allow",
    "Action": [
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage",
        "sqs:SendMessage"
    ],
    "Resource": "arn:aws:sqs:*:*:*",
    "Condition": {
        "Null": {
            "aws:ResourceTag/ManagedByPartnerDotCom": "false"
        },
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
}
```

Policy template to tag new queues on creation:

```
{
    "Effect": "Allow",
    "Action": [
        "sqs:CreateQueue",
        "sqs:TagQueue"
```

```
    ],
    "Resource": "arn:aws:sqs:*:*:*",
    "Condition": {
        "Null": {
            "aws:RequestTag/ManagedByPartnerDotCom": "false"
        }
    }
}
```

Policy template to tag pre-existing queues and create the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:TagQueue"
            ],
            "Resource": "arn:aws:sqs:*:@{AccountId}:@{QueueName}",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": "ManagedByPartnerDotCom"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:PutRolePermissionsBoundary",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::@{AccountId}:role/partner.com/SQSAccessor",
            "Condition": {
                "StringEquals": {
                    "iam:PermissionsBoundary": "arn:aws:iam::partner:policy/
permissions-boundary/partner.com/SQSAccessorBoundary_2025_01_15"
                }
            }
        }
    ]
}
```

This approach allows customers to explicitly confirm which specific resources can be accessed long-term.

Example 4: Updating the Permission Boundary

To update a permission boundary, register a new version with a new date suffix and request permission to replace it.

Updated permission boundary with additional permission:

```
{
    "Effect": "Allow",
    "Action": [
        "sqs:DeleteMessage",
        "sqs:PurgeQueue",
        "sqs:ReceiveMessage",
        "sqs:SendMessage"
    ],
    "Resource": "arn:aws:sqs:*:*:*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
}
```

As the second version, this policy has the ARN: arn:aws:iam::partner:policy/permissions-boundary/partner.com/SQSAccessorBoundary_2025_01_20

Policy template to update the permission boundary on the existing role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PutRolePermissionsBoundary"
            ],
            "Resource": "arn:aws:iam::@{AccountId}:role/partner.com/SQSAccessor",
            "Condition": {
                "StringEquals": {
```

```
                    "iam:PermissionsBoundary": "arn:aws:iam::partner:policy/
permissions-boundary/partner.com/SQSAccessorBoundary_2025_01_20"
                }
            }
        }
    ]
}
```

Customers must approve this delegation request to update the permission boundary on the existing role.

Building your integration

Understanding the Request Lifecycle

Before building your integration, it's important to understand how delegation requests progress from creation to completion.

Request States

A delegation request progresses through the following states:

State	Description
Unassigned	Request created but not yet associated with a customer account and an IAM principal. The request might have been created without specifying a target account, or with a target account ID but not yet claimed by the account owner.
Assigned	Request associated with a customer account and awaiting review
Pending Approval	Customer has forwarded the request to an administrator for approval
Accepted	Request approved by customer but exchange token not yet released
Finalized	Exchange token released to product provider. The delegation period (exchange token validity) begins when the request reaches Finalized state
Rejected	Request rejected by customer

State	Description
Expired	Request expired due to inactivity or timeout

State Transitions

Normal Flow (Approval Path)

- Unassigned → Assigned: Customer associates the request with their account
- Assigned → Accepted OR Assigned → Pending Approval: Customer approves the request directly OR forwards to administrator for review
- Pending Approval → Accepted: Administrator approves the request
- Accepted → Finalized: Customer releases the exchange token

Rejection Path

- Assigned → Rejected: Customer rejects the request
- Pending Approval → Rejected: Administrator rejects the request
- Accepted → Rejected: Customer revokes approval before releasing token

Expiration Path

Requests automatically expire if no action is taken within the specified timeframe:

- Unassigned → Expired (1 day)
- Assigned → Expired (7 days)
- Pending Approval → Expired (7 days)
- Accepted → Expired (7 days)
- Rejected → Expired (7 days)
- Finalized → Expired (7 days)

Terminal States

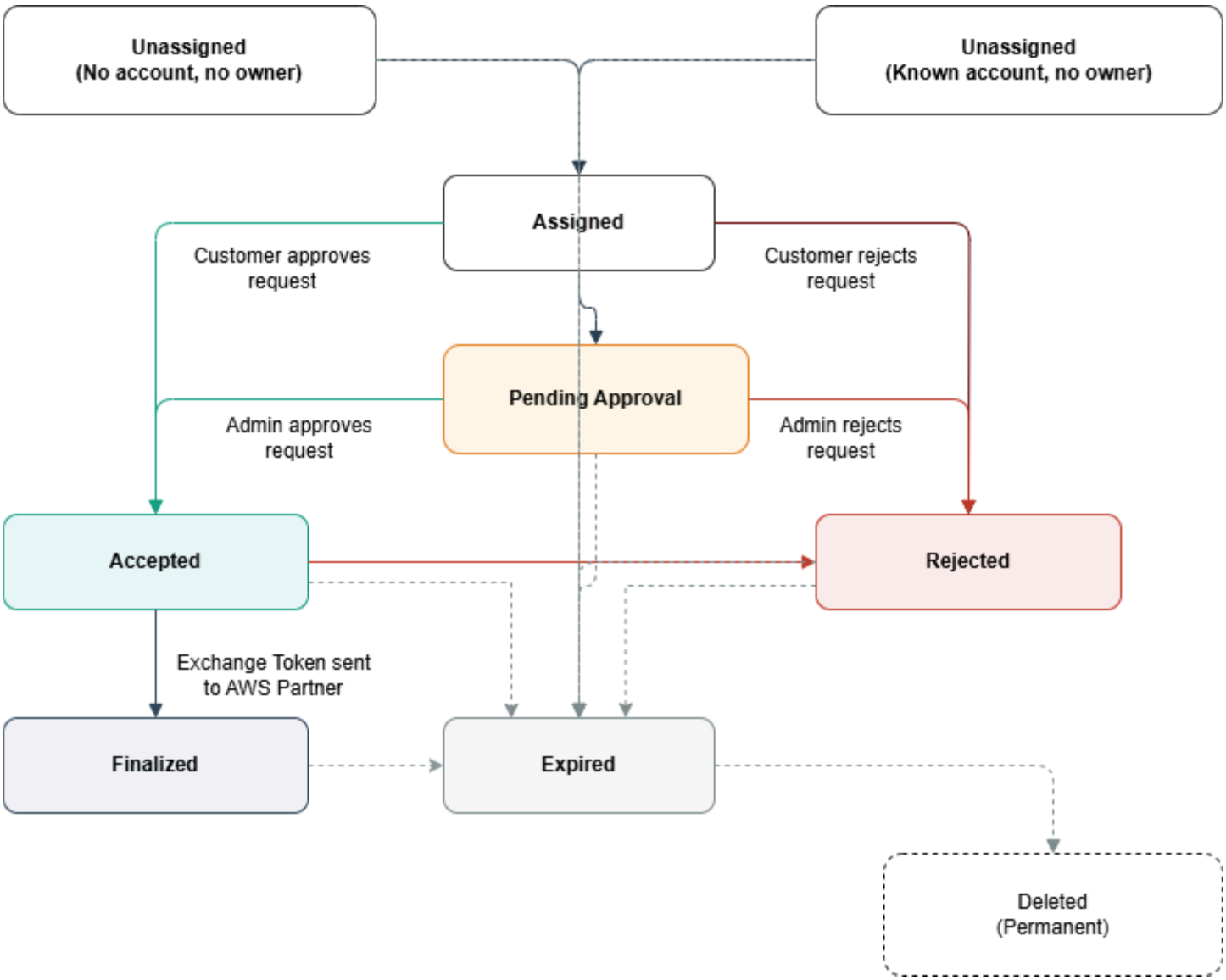
The following states are terminal (no further transitions):

- Finalized: Exchange Token sent
- Rejected: Request was denied
- Expired: Request timed out or delegation period ended

Expired requests are eventually deleted from the system after the retention period.

Managing delegation request states in your application

As a partner, you must track delegation request states in your system and surface them to your customers. When you receive SNS notifications about state changes, store these updates in your backend and reflect them in your customer-facing UI. Pay special attention to the Pending Approval state—when a customer forwards a request to an administrator for review, AWS sends a Pending Approval notification to you. Requests can remain in this state for up to 7 days while waiting for administrator action. During this time, show customers that their request is pending administrator approval in your application. Consider providing a deep link to the AWS Console where customers can check the request status or follow up with their administrator. Properly handling the state machine in your backend and surfacing the correct state information to customers at each stage is important for a good integration experience.



Configuring Notifications

IAM uses Amazon Simple Notification Service (SNS) to communicate delegation request state changes to you. When you create a delegation request, you must provide an SNS topic ARN from your registered AWS account. IAM will publish messages to this topic for important events, including when customers approve or reject requests and when the exchange token is ready.

> **Note**
>
> SNS topics cannot be in opt-in AWS Regions. Your SNS topic must be in an AWS Region that is enabled by default. For a list of opt-in Regions, see Managing AWS Regions in the AWS Account Management guide.

SNS Topic Configuration

To receive delegation request notifications, you must configure your SNS topic to grant IAM permissions to publish messages to it. Add the following policy statement to your SNS topic policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowIAMServiceToPublish",
            "Effect": "Allow",
            "Principal": {
                "Service": "iam.amazonaws.com"
            },
            "Action": "SNS:Publish",
            "Resource": "arn:aws:sns:us-east-1:111122223333:TOPIC-NAME"
        }
    ]
}
```

> **Important**
>
> The SNS topic must be in one of your registered AWS accounts. IAM will not accept SNS topics from other accounts. If the topic policy is not correctly configured, you will not receive state change notifications or the exchange token.

Notification Types

IAM sends two types of notifications:

StateChange Notifications

Sent when a delegation request transitions to a new state (Assigned, Pending Approval, Accepted, Finalized, Rejected, Expired).

ExchangeToken Notifications

Sent when a customer releases the delegation token (state Finalized). This notification includes the exchange token you need to obtain credentials.

Notification States

You will receive notifications for the following delegation request states:

State	Notification Type	Description
ASSIGNED	StateChange	Request has been associated with a customer account
PENDING APPROVAL	StateChange	Customer has forwarded the request to an administrator for approval
ACCEPTED	StateChange	Customer has approved the request but not yet released the token
FINALIZED	StateChange	Customer has released the exchange token
FINALIZED	ExchangeToken	This notification contains the Exchange Token
REJECTED	StateChange	Customer has rejected the request
EXPIRED	StateChange	Request expired before completion

Notification Message Format

IAM publishes standard SNS notifications. The delegation request information is contained in the Message field as a JSON string.

Common Fields (All Notifications)

Field	Type	Description
Type	String	Either "StateChange" or "ExchangeToken"
RequestId	String	The IAM delegation request ID
Requestor WorkflowId	String	The workflow ID you provided when creating the request
State	String	Current state of the request

Field	Type	Description
OwnerAccountId	String	Customer's AWS account ID
UpdatedAt	String	Timestamp when the state changed (ISO 8601 format)

Additional Fields (ExchangeToken Notifications Only)

Field	Type	Description
ExchangeToken	String	The token to exchange for credentials using AWS STS GetDelegatedAccessToken API
ExpiresAt	String	When the delegated access expires (ISO 8601 format)

Example Notifications

StateChange Notification

```
{
  "Type": "Notification",
  "MessageId": "61ee8ad4-6eec-56b5-8f3d-eba57556aa13",
  "TopicArn": "arn:aws:sns:us-east-1:123456789012:partner-notifications",
  "Message": "{\"RequestorWorkflowId\":\"workflow-12345\",\"Type\":\"StateChange\",
\"RequestId\":\"dr-abc123\",\"State\":\"ACCEPTED\",\"OwnerAccountId\":\"111122223333\",
\"UpdatedAt\":\"2025-01-15T10:30:00.123Z\"}",
  "Timestamp": "2025-01-15T10:30:00.456Z",
  "SignatureVersion": "1",
  "Signature": "...",
  "SigningCertURL": "...",
  "UnsubscribeURL": "..."
}
```

ExchangeToken Notification

```
{
  "Type": "Notification",
  "MessageId": "e44e5435-c72c-5333-aba3-354406782f5b",
  "TopicArn": "arn:aws:sns:us-east-1:123456789012:partner-notifications",
```

```
  "Message": "{\"RequestId\":\"dr-abc123\",\"RequestorWorkflowId\":\"workflow-12345\",
\"State\":\"FINALIZED\",\"OwnerAccountId\":\"111122223333\",\"ExchangeToken\":
\"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...\",\"ExpiresAt\":\"2025-01-15T18:30:00.123Z\",
\"UpdatedAt\":\"2025-01-15T10:30:00.456Z\",\"Type\":\"ExchangeToken\"}",
  "Timestamp": "2025-01-15T10:30:00.789Z",
  "SignatureVersion": "1",
  "Signature": "...",
  "SigningCertURL": "...",
  "UnsubscribeURL": "..."
}
```


Exchange Tokens

An exchange token or a trade in token is issued by IAM when a customer accepts and finalizes a delegation request. The product provider uses this exchange or trade in token to call the AWS STS GetDelegatedAccessToken API to obtain temporary AWS credentials with the permissions customers approved. The exchange token itself does not grant access to your AWS resources; it must be exchanged for actual credentials through AWS STS.

The exchange token can only be redeemed by the product provider account that created the delegation request. The requesting account is embedded in the token, ensuring that only the authorized product provider can obtain credentials to access customer account.

Access Duration

The delegation period starts when customer releases the exchange token, not when the product provider redeems it. After the customer releases the token:

- The product provider receives the token through an SNS notification
- They can immediately exchange it for credentials
- Credentials expire at: release time + approved duration
- The product provider can exchange the token multiple times before expiration to obtain fresh credentials if needed

Multiple Redemptions

Product providers can exchange the token multiple times during the validity period to obtain fresh credentials. However, all credentials obtained from the same exchange token expire at the same time, based on when you released the token.

Example: If you approve a 2-hour delegation request and release the token at 10:00 AM:

Token Release Time	Token Exchange Time	Credential Expiry	Usable Time
10:00 am	10:00 am	12:00 pm	2 hours
10:00 am	10:20 am	12:00 pm	1 hour 40 minutes
10:00 am	11:40 am	12:00 pm	20 minutes
10:00 am	12:10 pm	Failed (token expired)	0 minutes

As shown in the table, exchanging the token later in the validity period results in less usable time for the product provider.

Account access manager

What is account access manager?

Account access manager lets you assign AWS account access to users and groups in your IAM Identity Center organization instance. You assign access using the IAM roles in your AWS accounts. You can use account access manager alongside IAM Identity Center permission sets or on its own. Both account access manager and IAM Identity Center permission sets use the users and groups that you synchronized from your identity source or created directly in IAM Identity Center.

Why use account access manager?

Account access manager enables you to assign your existing IAM roles to the users and groups you have synchronized into IAM Identity Center from your corporate source of identities. Using your existing IAM roles gives you extra flexibility to tailor user access, including:

- Configure IAM role tags and use session tags for attribute-based access control (ABAC).
- Configure IAM role trust policy to restrict role assumption or use IdP-asserted claims as trust policy conditions.
- Configure IAM role paths to group roles logically and simplify administration. For more information, see Configure IAM role paths to group roles logically and simplify administration.

The following table compares the three account access options:

	IAM roles + account access manager	IAM Identity Center permission sets	IAM roles + IAM federation (direct role assumption)
Best for	Custom IAM roles that vary across accounts	Uniform baseline human access such as read-only and admin provisioned as immutable IAM roles	Suitable for workloads

	IAM roles + account access manager	IAM Identity Center permission sets	IAM roles + IAM federation (direct role assumption)
Scope	Multiple accounts (AWS organization)	Multiple accounts (AWS organization)	Per account
IAM role provisioning	Requires infrastructure as code (IaC) or manual provisioning through the AWS IAM console	Automatically provisioned	Requires infrastructure as code (IaC) or manual provisioning through AWS IAM and external IdP consoles
Role-to-identity mapping resides in	Account access manager	IAM Identity Center	External IdP
End user experience	Users access AWS accounts through the AWS account access portal URL after signing into AWS as IAM Identity Center users.	Users access AWS accounts through the AWS access portal after signing into AWS as IAM Identity Center users.	Users access AWS accounts through account-specific SAML applications (icons) in their IdP portal.

	IAM roles + account access manager	IAM Identity Center permission sets	IAM roles + IAM federation (direct role assumption)
AWS CLI user experience	1. User signs into AWS and then accesses an account with a specific role in a web browser. 2. User types **aws login** on the command line to retrieve the IAM role session credentials from the browser. For the duration of the IAM role session, the user can keep working with the AWS CLI without having to re-authenticate in step 1.	User configures a profile including the desired account and role pair by typing **aws configure sso**. User types **aws sso login** to initiate a new session using a profile. If there is no active IAM Identity Center session, the user authenticates in a browser and authorizes access there. See the AWS IAM Identity Center User Guide for more details.	Custom integrations using AWS SDK credential providers

Key capabilities

Account access portal

Account access manager provides users with an account access portal where they can view and access the AWS accounts assigned to them through account access manager.

CLI access

Your workforce can access AWS accounts through the AWS CLI using the **aws login** command after signing into AWS in a browser.

Account access APIs

Account access manager has its own API namespace, `account-access`, which is separate from the IAM API. You use this namespace in the AWS CLI and SDKs.

Getting started with account access manager

The following outlines how you can get started with account access manager. This assumes the IAM roles you want to assign already exist in their respective AWS accounts. If these roles do not exist, see Create a role using custom trust policies for information about how to create new roles.

To get started with account access manager

1. Make sure you have an organization instance of IAM Identity Center enabled and your workforce users and groups provisioned there. See the Getting started with IAM Identity Center section in the AWS IAM Identity Center User Guide for more information. See the section called "Prerequisites and considerations" later in this topic for more details.

   > **Note**
   >
   > If you are using IAM Identity Center replicated to multiple AWS Regions or with a customer managed key (CMK), you must add `kms:Decrypt` permission for account access manager to use this CMK when connecting to IAM Identity Center. For more information, see Baseline KMS key policy in the *AWS IAM Identity Center User Guide*.

2. Make sure you have permissions to enable account access manager and manage account assignments in it. For more information, see the section called "Identity and access management for account access manager".
3. Enable account access manager through the console or API.
4. Update your firewalls and gateways for access to the account access portal.
5. Grant account access manager access to IAM roles by updating their trust policy.
6. Start assigning account access to your workforce users and groups from IAM Identity Center.
7. Let your users know how to access AWS accounts using account access manager role assignments.

Prerequisites and considerations

The following prerequisites and considerations apply:

- Account access manager is available in AWS Commercial Regions enabled by default. Opt-in Commercial Regions, GovCloud (US), China, and other non-commercial Regions are not supported.
- At enablement, account access manager connects to a single organization instance of IAM Identity Center. You must enable an AWS organization and an organization instance of IAM Identity Center beforehand. Account instances of IAM Identity Center are not supported.
- We recommend enabling account access manager in the same AWS Region where you enabled your IAM Identity Center instance. If you haven't done it already, see Considerations for choosing an AWS Region in the AWS IAM Identity Center User Guide.
- Both the organization instance of IAM Identity Center and account access manager must be enabled in your organization's management account.

Enable account access manager

After creating an organization instance of IAM Identity Center, you can enable account access manager through its console or API. You must do this in the same AWS account and Region as your IAM Identity Center instance.

Before enabling account access manager, run the following command from the AWS organization management account to grant account access manager access to your organization:

```
aws organizations enable-aws-service-access --service-principal account-
access.amazonaws.com
```

> **Note**
>
> While the account access manager console is part of the IAM console, its API lives in a separate namespace.

Use one of the following methods to enable account access manager:

Console

To enable account access manager

1. Sign in to your organization's management account.
2. In the AWS Identity and Access Management console navigation pane, choose **Account access manager**.
3. Confirm the console Region is set to the primary Region of your IAM Identity Center instance.
4. If there is no organization instance of IAM Identity Center, you will be prompted to enable one now before continuing. Otherwise, choose **Enable account access manager**.

AWS CLI

First find the ARN of the IAM Identity Center instance and confirm the Region code matches the primary Region of IAM Identity Center. Then, run the following command:

```
aws account-access create-application \
  --region <Region> \
  --identity-source '{
    "identityCenter": {
      "instanceArn": "<IAM_IDENTITY_CENTER_INSTANCE_ARN>"
    }
  }'
```

Update firewalls and gateways

The account access portal provides your users with single sign-on access to the AWS accounts assigned to them through account access manager.

If you filter access to specific AWS domains or URL endpoints by using a web content filtering solution such as next-generation firewalls (NGFW) or Secure Web Gateways (SWG), you must allowlist the domains and URL endpoints associated with the account access portal.

The following list provides the dual-stack domains and URL endpoints to add to your web-content filtering solution allowlists. Account access manager has no IPv4-only endpoints.

Dual-stack allow list

The account access portal URL:

- `https://[Tenant-ID].account-access.[Region].app.aws`

The APIs that the account access portal calls:

- `[Tenant-ID].account-access.[Region].app.aws/api/*`
- `[Tenant-ID].account-access.[Region].app.aws/auth/*`

Tenant-ID is a unique random identifier associated with account access manager. To find its value, navigate to the **Settings** tab in the account access manager console, and look up the **Application URL** in the **Instance details** section. Tenant-ID is the first subdomain in the URL.

The account access manager service endpoints:

- Control-plane: `account-access.[Region].api.aws`
- Runtime: `account-access-runtime.[Region].api.aws`
- Console: `console.aws.amazon.com/account-access`

If you use the AWS CLI, you can find the tenant ID by running the following CLI command in your organization's management or delegated administrator account.

```
aws account-access list-applications
```

The response contains information about account access manager including the application ARN and the tenant ID.

```
{
    "applications": [
        {
            "applicationArn": "arn:aws:account-access:us-
west-2:123456789012:application/1234567890abcdef",
            "tenantId": "aa-gyxmap389",
            "createdAt": "2026-03-27T18:31:19+00:00",
            "updatedAt": "2026-03-27T18:31:19+00:00"
        }
    ]
}
```

Considerations for allowlisting domains and URL endpoints

In addition to the allowlist requirements for the account access portal, the other services and applications you use might require allowlisting of domains.

- You need to allowlist the AWS access portal and its dependencies. For more information, see Updating firewalls and gateways to allow access to the AWS access portal in the AWS IAM Identity Center User Guide.
- To access AWS accounts, the AWS Management Console, and the account access manager console, you must allowlist additional domains. Refer to Troubleshooting in the *AWS Management Console Getting Started Guide* for a list of AWS Management Console domains.

Delete account access manager

When account access manager is deleted, all the data in it is deleted and cannot be recovered.

Use the following procedure to delete account access manager.

Console

To delete your account access manager instance

1. Sign in to your organization's management account.
2. In the AWS Identity and Access Management console navigation pane, choose **Account access manager**.
3. Confirm the console Region is set to the primary Region of your account access manager instance.
4. Choose the **Settings** tab.
5. In the **Delete account access manager** section, choose **Delete**.
6. The dialog box prompts you to confirm. When ready, choose **Delete**.

AWS CLI

Run the following command to delete account access manager:

```
aws account-access delete-application \
  --region "<Region>" \
```

```
  --application-arn "<account_access_manager_ARN>"
```

Warning

Deleting account access manager is irreversible and has the following consequences:

- All role assignments are permanently removed. Your users immediately lose the ability to access AWS accounts with IAM roles assigned through account access manager.
- Account access manager is removed from the AWS access portal for all users.
- Active sessions of IAM roles assigned through account access manager continue until they expire, but users cannot start new sessions.
- If you re-enable account access manager later, you must recreate all assignments from scratch. Previous assignments are not recoverable.

Before deleting, consider disabling user access in individual accounts to temporarily restrict access.

Manage access to AWS accounts

Account access manager is integrated with IAM Identity Center and AWS Organizations to centrally manage the assignment of existing IAM roles across multiple AWS accounts without configuring each of your accounts individually. With account access manager, you assign existing IAM roles in your AWS accounts to IAM Identity Center users and groups to control their access to specific AWS accounts.

With account access manager, your teams can create their own custom IAM roles in their AWS accounts, and you as administrator use account access manager to centrally manage assignments of IAM Identity Center users and groups to these IAM roles.

Note

Within the context of account access manager, the terms *users* and *groups* exclusively refer to workforce *users* and *groups* in IAM Identity Center.

Delegated administration

When you enable account access manager, AWS creates it in the management account in AWS Organizations because account access manager manages IAM role assignments for an entire AWS Organization. You can choose to delegate administration of account access manager to a member account in AWS Organizations.

If you choose to register a member account as a delegated administrator for account access manager, users in this account can perform most account access manager administrative tasks. The following table indicates whether an account access manager administrative task can be performed by users in an organization management account, a delegated administration member account, or another member account. IAM role management is excluded from the list of administrative tasks because it takes place outside of account access manager.

Account access manager administrative tasks	Management account	Delegated administrator account	Member account
Manage IAM role assignments*	Yes	Yes	No
Enable account access manager	Yes	No	No
Delete account access manager	Yes	No	No
Enable or disable user access in the management account	Yes	No	No
Enable or disable user access in a member account	Yes	Yes	No
Register or deregister a member account as	Yes	No	No

Account access manager administrative tasks	Management account	Delegated administrator account	Member account
a delegated administrator			

*Refer to the best practices for delegated administration regarding user and group assignments to the management account.

Register a member account

To configure delegated administration, you must first register a member account in your organization as a delegated administrator. Users in that member account who have sufficient permissions will have administrative access to account access manager.

Account access manager supports registering only one member account as a delegated administrator at a time. You can only register a member account while signed in with credentials from the management account.

Use the following procedure to register a member account in your AWS organization as a delegated administrator.

To register a member account

1. Sign in to the AWS Management Console using the credentials of your management account in AWS Organizations. Management account credentials are required to run the RegisterDelegatedAdministrator API.
2. Select the Region where account access manager was enabled and then open the AWS Identity and Access Management console.
3. Choose **Account access manager**, and then choose the **Settings** tab.
4. In the **Delegated administrator** section, choose **Register**.
5. On the **Register delegated administrator** page, select the AWS account you want to register, and then choose **Register**.

Deregister a member account

You can only deregister a member account while signed in with credentials from the management account.

Use the following procedure to deregister a member account in your AWS organization that had previously been designated as a delegated administrator.

> **Important**
>
> When you deregister an account, you remove the ability for all admin users to manage account access manager from that account. They can no longer administer AWS account access from this account.
> This operation does not affect permissions or assignments configured in account access manager. Your end users continue to access their AWS accounts from the account access portal.

To deregister a member account

1. Sign in to the AWS Management Console using the credentials of your management account in AWS Organizations. Management account credentials are required to run the DeregisterDelegatedAdministrator API.
2. Select the Region where account access manager was enabled, and then open the AWS Identity and Access Management console.
3. Choose **Account access manager**, and then choose the **Settings** tab.
4. In the **Delegated administrator** section, choose **Deregister account**.
5. In the **Deregister delegated administrator** dialog box, review the security implications, and then confirm that you understand.
6. Choose **Deregister**.

View the registered delegated administrator account

Use the following procedure to find which member account in your AWS Organizations has been configured as the delegated administrator for account access manager.

To view your registered member account

1. Sign in to the AWS Management Console using the credentials of your management account in AWS Organizations.
2. Select the Region where account access manager was enabled and then open the AWS Identity and Access Management console.
3. Choose **Account access manager**, and then choose the **Settings** tab.
4. In the **Delegated administrator** section, you can view the current delegated administrator account details.

Prepare your IAM roles

Before you start assigning IAM roles to IAM Identity Center users and groups, you need to:

1. Identify the relevant IAM roles in the accounts of interest, and
2. Make sure the trust policy of the IAM roles contains the Statement provided below.

If the role has an existing trust policy, add this statement to its Statement array. If you're creating a new role, you can use the entire policy document. This policy statement allows account access manager to assume the role and set context on behalf of users. The `Condition` block provides confused-deputy protection by ensuring that account access manager can assume the role only on behalf of your own account and your account access manager application. For more information, see Update a role trust policy.

Replace `<your-account-id>` with your AWS account ID, `<region>` with your IAM Identity Center primary Region, and `<application-id>` with your account access manager application ID. You can find the application ARN on the **Settings** page in the account access manager console.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccountAccessManagerIAMRoleTrustPolicyStatement",
      "Effect": "Allow",
      "Principal": {
        "Service": "account-access.amazonaws.com"
      },
      "Action": [
```

```
        "sts:AssumeRole",
        "sts:SetContext"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<your-account-id>",
          "aws:SourceArn": "arn:aws:account-access:<region>:<your-account-
id>:application/<application-id>"
        }
      }
    }
  ]
}
```

Optional — `sts:TagSession`. Add `sts:TagSession` to the `Action` list if you want upstream assertion attributes from your identity provider (for example, group memberships, department, cost-center) to propagate as principal tags on the session that account access manager issues. Without this action, role assumption still succeeds; only session-tag propagation is suppressed. Add it when downstream IAM policies in the target account use principal tags (`aws:PrincipalTag/<key>`); leave it out otherwise. Keep the same `Condition` block.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccountAccessManagerIAMRoleTrustPolicyStatement",
      "Effect": "Allow",
      "Principal": {
        "Service": "account-access.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:SetContext",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<your-account-id>",
          "aws:SourceArn": "arn:aws:account-access:<region>:<your-account-
id>:application/<application-id>"
        }
      }
```

```
    }
  ]
}
```

> **Note**
>
> We recommend you update your IAM role creation workflows to add this trust policy statement automatically where needed.

Assign and remove access

Assign user or group access to an AWS account

You can assign a user or group access to an AWS account by creating an assignment between the user or group, the account, and the IAM role that the user or group can assume in the account. Thus, a single assignment is a triplet consisting of the user or group ID, the account ID, and the IAM role ARN.

If you want to perform bulk assignments such as assigning the same role to a number of users and groups, you must create each assignment individually. You can script this procedure using the account access API, and the AWS SDK/CLI.

Console

The console provides a few different ways to assign new access depending on whether you want to first identify the account, the user, or the group.

> **Note**
>
> The account access manager console provides a search capability for users, groups, and accounts. To identify the role you want to assign, you must find its name (or ARN) beforehand. The account access manager console cannot detect IAM roles in AWS accounts.

For all procedures in this section, sign in to the management account (or delegated admin account), select the Region where account access manager is enabled, and navigate to **Account access manager** in the IAM console.

Accounts tab

To assign user or group access from the Accounts tab

1. Choose the **Accounts** tab.
2. If you already know the account ID, choose **Assign new access**. On the **Assign account access** page, look up the desired user or group, enter the account ID, and IAM role name, and then choose **Assign access**.
3. If you need to first find the account, you can search for the desired AWS account by its name, ID, or email address. Alternatively, you can review the organizational structure or account list to find the relevant AWS account. Choose the radio button in front of the account and then **Assign new access**. On the **Assign account access** page, look up the desired user or group, enter the IAM role name, and then choose **Assign access**.
4. The organizational structure or account list lets you choose an account to view account details and account access assignments. You can choose **Assign new access** from there too.

Users tab

To assign user access from the Users tab

1. Choose the **Users** tab.
2. You can search users by username and display name or find them in the displayed user list.
3. Choose the user you want to assign new access to.
4. The next page shows the user attributes, account assignments, and group memberships.
5. Choose **Assign new access**.
6. On the **Assign account access** page, enter the account ID and IAM role name, and then choose **Assign access**.

Groups tab

To assign group access from the Groups tab

1. In the navigation pane, choose **Account access manager**.
2. Choose the **Groups** tab.
3. You can search groups by group name or find them in the displayed group list.
4. Choose the group you want to assign new access to.

5. The next page shows the group attributes, account assignments, and user members.
6. Choose **Assign new access**.
7. On the **Assign account access** page, enter the account ID and IAM role name, and then choose **Assign access**.

AWS CLI

The following AWS CLI command can be used to create an account assignment for a user:

```
aws account-access create-entitlement \
  --region <Region> \
  --application-arn "<Account_access_manager_ARN>" \
  --entitlement '{
    "principalRole": {
      "principal": {
        "identityCenter": {
          "userId": "<Identity_Center_User_ID>"
        }
      },
      "roleArn": "arn:aws:iam::<ACCOUNT_ID>:role/<ROLE_NAME>"
    }
  }'
```

The following AWS CLI command can be used to create an account assignment for a group:

```
aws account-access create-entitlement \
  --region <Region> \
  --application-arn "<Account_access_manager_ARN>" \
  --entitlement '{
    "principalRole": {
      "principal": {
        "identityCenter": {
          "groupId": "<IDC_GROUP_ID>"
        }
      },
      "roleArn": "arn:aws:iam::<ACCOUNT_ID>:role/<ROLE_NAME>"
    }
  }'
```

Remove user or group access to an AWS account

You can find the assignment to remove in several ways.

Console

For all procedures in this section, sign in to the management account (or delegated admin account), select the Region where account access manager is enabled, and navigate to **Account access manager** in the IAM console.

Accounts tab

To remove user or group access from the Accounts tab

1. Choose the **Accounts** tab.
2. Find the relevant account and choose its name.
3. In the **Account access assignments** section, choose the user or group you want to remove access for, and choose **Unassign**.
4. In the modal, choose **Unassign** to confirm.

Users tab

To remove user access from the Users tab

1. Choose the **Users** tab.
2. Choose the user you want to remove access for.
3. In the **Account access assignments** section, choose the account you want to remove access for, and choose **Unassign**.
4. In the modal, choose **Unassign** to confirm.

AWS CLI

To remove access with the CLI, you first need to find the account assignment (entitlement) and then use its ID in the following command. You can see later in this topic how to search for account assignments for a specific account, user, group, or role. The CLI refers to assignments as entitlements.

```
aws account-access delete-entitlement \
```

```
    --region <Region> \
    --application-arn "<Account_access_manager_ARN>" \
    --entitlement-id "<ENTITLEMENT_ID>"
```

View or search account assignments

You can view account assignments in several ways.

Console

For all procedures in this section, sign in to the management account (or delegated admin account), select the Region where account access manager is enabled, and navigate to Account access manager in the IAM console.

Accounts tab

View user or group access from the Accounts tab

1. Choose the **Accounts** tab.
2. By default, you see the Hierarchy view. You can switch to the List view if desired. You can search accounts by their name, ID, email, and OU ID. When you find the desired account, choose its name.
3. On the account page, you can see the account assignments in the **Account access assignments** section. You can filter them further by username, user ID, group name, group ID, and IAM role ARN.

Users tab

View user access from the Users tab

1. Choose the **Users** tab.
2. Search for the user by name or user ID. When you find the desired user, choose the username.
3. On the user page, you can see the user's account assignments. You can filter them further by account ID, account name, and IAM role ARN.

Groups tab

View group access from the Groups tab

1. Choose the **Groups** tab.
2. Search for the group by name or group ID. When you find the desired group, choose the group name.
3. On the group page, you can see the group's account assignments. You can filter them further by account ID, account name, and IAM role ARN.

AWS CLI

The following commands illustrate how to retrieve account assignments using different filters. The CLI refers to assignments as entitlements.

Retrieve first 100 assignments (entitlements) for a specific account

```
aws account-access list-entitlements \
    --region <Region> \
    --application-arn "<Account_access_manager_ARN>" \
    --filter '{
      "principalRole": {
        "account": "<ACCOUNT_ID>"
      }
    }' \
    --max-results 100
```

Retrieve all assignments (entitlements) for a specific user

```
aws account-access list-entitlements \
    --region <Region> \
    --application-arn "<Account_access_manager_ARN>" \
    --filter '{
      "principal": {
        "type": "USER",
        "id": "<USER_ID>"
      }
    }'
```

Retrieve all assignments (entitlements) for a specific group

```
aws account-access list-entitlements \
    --region <Region> \
    --application-arn "<Account_access_manager_ARN>" \
    --filter '{
      "principal": {
        "type": "GROUP",
        "id": "<GROUP_ID>"
      }
    }'
```

Sessions and access control

Understanding sessions

Users who access AWS accounts using assignments in account access manager initiate the following session types:

1. **IAM role session** - The IAM role a user assumes is a temporary credential for access to AWS resources with a default duration of 1 hour.
2. **Account access portal session** – Users initiate this session when using the account access portal. It is managed as an application session by IAM Identity Center.
3. **AWS access portal session (also called user interactive session)** – This session is managed by IAM Identity Center.
4. **External identity source (IdP or Microsoft AD) session** – This session is applicable when the IAM Identity Center instance is connected to an external identity source for authentication.

For more information about user interactive and application sessions in IAM Identity Center including session duration, see Understanding authentication sessions in IAM Identity Center in the *AWS IAM Identity Center User Guide*.

Revoke user access

To revoke user access assumed through account access manager role assignments, see Revoke user access in the *AWS IAM Identity Center User Guide*.

Attribute-based access control

You can configure your identity provider (IdP) to send attributes as session tags through SAML assertions by setting the attribute name to `https://aws.amazon.com/SAML/Attributes/`

`AccessControl:TagKey`, where *`TagKey`* is the session tag key you want to populate. IAM Identity Center passes the attribute name and value from the IdP through for policy evaluation.

Workforce access

Account access manager enables you to assign AWS account access to your workforce users and groups in IAM Identity Center by mapping them to IAM roles in your AWS accounts. If you are new to IAM Identity Center, see Set up workforce access to AWS resources in the *AWS IAM Identity Center User Guide* for information about authentication sessions, user access management, password management, and multi-factor authentication.

User portals

To access the AWS accounts assigned to them, your workforce users can navigate directly to the account access portal using a bookmark, or launch it from within the AWS access portal.

Your users might interact with two user portals:

Account access portal

This is a portal (URL) that allows workforce users to access the AWS accounts assigned to them using account access manager. This portal (URL) can be bookmarked for direct access and is also available in the AWS access portal as an AWS managed application.

AWS access portal

This is the main entry point into AWS for IAM Identity Center users. This portal displays all AWS applications that users have access to, including the account access portal.

Tip

We recommend bookmarking both portals for faster navigation. If you use an external identity provider, you can also create bookmark apps for both portals there.

Getting started with the account access portal

Finding the account access portal URL

To find the account access portal URL, choose the **Settings** tab in the account access manager console. Look up the **Application URL** in the **Instance details** section. The URL follows the pattern `https://[Tenant-ID].account-access.[Region].app.aws`. Tenant-ID is a unique random identifier associated with account access manager.

If you use the AWS CLI, you can find the tenant ID by running the following CLI command in your organization's management or delegated administrator account.

```
aws account-access list-applications
```

The response contains information about account access manager including the application ARN and the tenant ID. Use the tenant ID and the Region code of account access manager to construct the application URL. Follow the pattern provided earlier in this section.

```
{
    "applications": [
        {
            "applicationArn": "arn:aws:account-access:us-
west-2:123456789012:application/1234567890abcdef",
            "tenantId": "aa-gyxmap389",
            "createdAt": "2026-03-27T18:31:19+00:00",
            "updatedAt": "2026-03-27T18:31:19+00:00"
        }
    ]
}
```

Setting up a vanity portal URL

For instructions on implementing custom vanity domains, see Regional routing for AWS access portals: Implementing custom vanity domains for IAM Identity Center on the AWS Security Blog. You can adapt these instructions to the URL pattern of the account access portal.

Communicating with your users

Communicate the following to your workforce users:

- The account access portal URL.

- The account access portal is available to your users also as an application in the **Applications** tab in the AWS access portal.
- If you enabled launching the account access portal from an external identity provider portal or another portal outside AWS, include this information in the communication.

User access to AWS accounts

This topic describes how users access the AWS accounts assigned to them with account access manager.

Getting your portal URL

You can access your assigned AWS accounts in several ways:

1. Your administrator might provide you with an account access portal URL (for example, `https://aa-gyxmap389.account-access.us-east-2.app.aws` or a vanity URL).
2. Your administrator might provide you with the URL for the main AWS access portal which displays all AWS applications assigned to you, including the account access portal.
3. Your organization might have enabled you to launch the account access portal from an external identity provider portal or another portal outside AWS.

Access AWS accounts with a web browser

The account access portal displays the AWS accounts you have access to, and the IAM roles you can assume in each account.

To access an account in a browser

1. Navigate to the account access portal URL and when prompted, sign in using your credentials as instructed by your administrator.

   > **Note**
   >
   > To launch the account access portal from the AWS access portal instead, navigate to the **Applications** tab.

2. The browser page displays the AWS accounts you are assigned to. You can search accounts by account name or ID.

3. Expand the account of interest to see all the roles you can assume in the account.
4. Choose the IAM role to sign in with (for example, Developer). A new browser tab opens where you are signed into the account with the chosen role.

Access AWS accounts with the AWS CLI or AWS SDKs

You can access AWS services programmatically by using the AWS Command Line Interface or AWS Software Development Kits (SDKs) with user credentials from IAM Identity Center.

The account access portal provides users with single sign-on access to their AWS accounts for account assignments managed in account access manager. After you sign in to the account access portal, you can get temporary credentials. You can then use the credentials in the AWS CLI or AWS SDKs to access resources in an AWS account.

If you're using the AWS CLI to access AWS services programmatically, you can use the procedures in this topic to initiate access to the AWS CLI. For information about the AWS CLI, see the *AWS Command Line Interface User Guide*.

If you're using the AWS SDKs to access AWS services programmatically, following the procedures in this topic also directly establishes authentication for the AWS SDKs. For information about the AWS SDKs, see the *AWS SDKs and Tools Reference Guide*.

AWS account access with aws login (recommended)

You can use the **aws login** command to gain access to AWS accounts you are assigned to in account access manager. First, install the latest AWS CLI. Then, follow this procedure:

To access an account with AWS CLI

1. In your default browser, sign in to your account access portal, and from there to the target AWS account using one of the assigned IAM roles for the account. Follow the procedure earlier in this topic.
2. On the command line, run the **aws login** command.
3. If you have not set a default Region, the AWS CLI prompts you to specify the AWS Region of your choice (for example, us-east-2). The AWS CLI will remember your choice for future use.
4. The AWS CLI opens a web page in your default browser and prompts you to continue with the active session you just initiated or sign in to a new session.

5. On the web page, choose the tile with the session you initiated earlier in the browser.
6. The browser page confirms that your credentials have been shared successfully and can be used until your session expires. You can now close the tab with the web page and continue working with the AWS CLI using this session.

> **Note**
>
> IAM Identity Center-specific AWS CLI commands such as **aws sso configure** and **aws sso login** don't work with account assignments in account access manager.

AWS account access with access keys

You can use the manual credential refresh method to get temporary credentials for an IAM role assigned to you in a specific AWS account. To do so, you copy and paste the required commands for the temporary credentials. With this method, you must refresh the temporary credentials manually.

You can run AWS CLI commands until your temporary credentials expire.

To get credentials that you manually refresh

1. Sign in to the account access portal as explained in Access AWS accounts with a web browser.
2. Locate the AWS account from which you want to retrieve access credentials and expand it to show the IAM role name (for example Developer).
3. Choose **Generate access key** next to the desired role.

> **Note**
>
> If you do not see any AWS account assignments, it is likely that you've not yet been assigned to any account. In this case, contact your administrator and ask them to assign access to you.

4. In the **Get credentials** dialog box, choose the **MacOS and Linux**, **Windows**, or **PowerShell** tab depending on the operating system on which you installed the AWS CLI.
5. Choose any of the following options:

   - **Option 1: Set AWS environment variables**

Choose this option to override all credential settings, including any settings in the credentials files and config files. For more information, see Environment variables to configure the AWS CLI in the *AWS Command Line Interface User Guide*.

To use this option, copy the commands to your clipboard, paste the commands into your AWS CLI terminal window, and then press Enter to set the required environment variables.

- **Option 2: Use individual values in your AWS service client**

  Choose this option to access AWS resources from an AWS service client. For more information, see Tools to Build on AWS.

  To use this option, copy the values to your clipboard, paste the values into your code, and assign them to the appropriate variables for your SDK. For more information, see the documentation for your specific SDK API.

Creating shortcut links to AWS Management Console destinations

The account access portal offers an option to create a shareable deep link.

To create a shortcut link

1. While signed into the account access portal, choose the **Create shortcut link** button.
2. In the dialog box:

   a. Choose an AWS account using the account ID or account name. As you type, a drop-down menu displays matching account IDs and names that you can access. You can choose only an account to which you have access.

   b. (Optional) Choose an IAM role from the drop-down list. These are the IAM roles assigned to you for the selected account. If you omit choosing the role, users are prompted to select one assigned to them for the chosen account when using the shortcut link.

   > **Note**
   >
   > You cannot grant new access with shortcut links. Shortcut links work only with the accounts and roles already assigned to the user.

Constructing secure AWS Management Console shortcut links with URL encoding

All parameter values of the URL, including the account ID, IAM role name, and destination URL, must be URL-encoded.

Shortcut links extend the account access portal URL with the following path:

```
/#/console?account_id=[account_ID]&role_name=[role_name]&destination=[destination_URL]
```

The full URL in a commercial AWS Region follows this pattern:

```
https://[Tenant-ID].account-access.[Region].app.aws/#/console?
account_id=[account_ID]&role_name=[role_name]&destination=[destination_URL]
```

Here's an example shortcut link that signs a user into account 123456789012 with the S3FullAccess IAM role name, and takes them to the S3 console home page:

```
https://aa-gyxmap389.account-access.us-east-2.app.aws/#/console?
account_id=123456789012&role_name=S3FullAccess&destination=https%3A%2F
%2Fconsole.aws.amazon.com%2Fs3%2Fhome
```

Considerations

The following considerations apply to the use of shortcut links:

- If you share shortcut links with coworkers, they need to have the same account assignments to be able to use the links in the same way.
- Shortcut links are Regional links. If the referenced Region is experiencing a disruption, the link might not work.
- Shortcut links contain the account ID and role name in plain text. Consider whether your organization's policies restrict where this metadata is stored.

Security in account access manager

This section provides security information specific to account access manager. It complements the information in *Security*, which covers security topics applicable to IAM as a whole.

Identity and access management for account access manager

The following sections describe the permissions you need to:

- Administer account access manager in your organization management account
- Administer account access manager from a delegated administrator account

Administering account access manager from your organization management account

To administer account access manager settings in the AWS organization management account you need the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AAMCreateApplication",
      "Effect": "Allow",
      "Action": "account-access:CreateApplication",
      "Resource": "*"
    },
    {
      "Sid": "AAMApplicationAndEntitlementManagement",
      "Effect": "Allow",
      "Action": [
        "account-access:CreateEntitlement",
        "account-access:DeleteApplication",
        "account-access:DeleteEntitlement",
        "account-access:GetApplication",
        "account-access:GetEntitlement",
        "account-access:ListEntitlements",
        "account-access:ListTagsForResource",
        "account-access:TagResource",
        "account-access:UntagResource"
      ],
      "Resource": "arn:aws:account-access:*:111122223333:application/*"
    },
    {
      "Sid": "AAMListApplications",
      "Effect": "Allow",
```

```
      "Action": "account-access:ListApplications",
      "Resource": "*"
    },
    {
      "Sid": "DependentIdentityCenterActions",
      "Effect": "Allow",
      "Action": [
        "sso:CreateApplication",
        "sso:DeleteApplication",
        "sso:DescribeApplication",
        "sso:DescribeInstance",
        "sso:DescribeRegion",
        "sso:ListInstances",
        "sso:PutApplicationAccessScope",
        "sso:PutApplicationAssignmentConfiguration",
        "sso:PutApplicationAuthenticationMethod",
        "sso:PutApplicationGrant"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DependentIdentityStoreActions",
      "Effect": "Allow",
      "Action": [
        "identitystore:DescribeGroup",
        "identitystore:DescribeUser",
        "identitystore:ListGroupMemberships",
        "sso-directory:DescribeGroup",
        "sso-directory:DescribeGroups",
        "sso-directory:DescribeUser",
        "sso-directory:DescribeUsers",
        "sso-directory:ListGroupsForUser",
        "sso-directory:ListMembersInGroup",
        "sso-directory:SearchGroups",
        "sso-directory:SearchUsers"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DependentOrganizationsActions",
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeAccount",
        "organizations:DescribeOrganization",
```

```
        "organizations:ListAccounts",
        "organizations:ListAccountsForParent",
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:ListDelegatedAdministrators",
        "organizations:ListOrganizationalUnitsForParent",
        "organizations:ListRoots"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DependentKMSActions",
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "*"
    },
    {
      "Sid": "CreateServiceLinkedRole",
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::111122223333:role/aws-service-role/account-
access.amazonaws.com/AWSServiceRoleForAccountAccessManager",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "account-access.amazonaws.com"
        }
      }
    }
  ]
}
```

Optional: To constrain `account-access:CreateApplication` by tag, replace the AAMCreateApplication statement with:

```
{
  "Sid": "AAMCreateApplicationConstrainedByTag",
  "Effect": "Allow",
  "Action": "account-access:CreateApplication",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/Environment": "Production"
    },
    "ForAllValues:StringEquals": {
```

```
      "aws:TagKeys": [
        "Environment"
      ]
    }
  }
}
```

Optional: To manage and register delegated administrators for account access manager, add:

```
{
  "Sid": "ManageDelegatedAdministrator",
  "Effect": "Allow",
  "Action": [
    "organizations:DeregisterDelegatedAdministrator",
    "organizations:RegisterDelegatedAdministrator"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "organizations:ServicePrincipal": "account-access.amazonaws.com"
    }
  }
}
```

To assign IAM roles to your workforce users and groups from the AWS organization management account without administering account access manager settings you need the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AAMManageEntitlementsOnly",
      "Effect": "Allow",
      "Action": [
        "account-access:CreateEntitlement",
        "account-access:DeleteEntitlement",
        "account-access:GetApplication",
        "account-access:GetEntitlement",
        "account-access:ListEntitlements"
      ],
      "Resource": "arn:aws:account-access:*:111122223333:application/*"
    },
```

```
    {
      "Sid": "AAMListApplications",
      "Effect": "Allow",
      "Action": "account-access:ListApplications",
      "Resource": "*"
    },
    {
      "Sid": "DependentIdentityCenterActions",
      "Effect": "Allow",
      "Action": "sso:ListInstances",
      "Resource": "*"
    },
    {
      "Sid": "DependentIdentityStoreActions",
      "Effect": "Allow",
      "Action": [
        "identitystore:DescribeGroup",
        "identitystore:DescribeUser",
        "identitystore:ListGroupMemberships",
        "sso-directory:DescribeGroup",
        "sso-directory:DescribeGroups",
        "sso-directory:DescribeUser",
        "sso-directory:DescribeUsers",
        "sso-directory:ListGroupsForUser",
        "sso-directory:ListMembersInGroup",
        "sso-directory:SearchGroups",
        "sso-directory:SearchUsers"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DependentOrganizationsActions",
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeAccount",
        "organizations:DescribeOrganization",
        "organizations:ListAccounts",
        "organizations:ListAccountsForParent",
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:ListDelegatedAdministrators",
        "organizations:ListOrganizationalUnitsForParent",
        "organizations:ListRoots"
      ],
      "Resource": "*"
```

```
    }
  ]
}
```

Administering account access manager in a delegated administration account

To administer account access manager settings from a delegated administration account you need the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AAMDelegatedAdminManagement",
      "Effect": "Allow",
      "Action": [
        "account-access:CreateEntitlement",
        "account-access:DeleteEntitlement",
        "account-access:GetApplication",
        "account-access:GetEntitlement",
        "account-access:ListEntitlements",
        "account-access:ListTagsForResource",
        "account-access:TagResource",
        "account-access:UntagResource"
      ],
      "Resource": "arn:aws:account-access:*:111122223333:application/*"
    },
    {
      "Sid": "AAMListApplications",
      "Effect": "Allow",
      "Action": "account-access:ListApplications",
      "Resource": "*"
    },
    {
      "Sid": "DependentIdentityCenterActions",
      "Effect": "Allow",
      "Action": [
        "sso:DescribeApplication",
        "sso:DescribeInstance",
        "sso:DescribeRegion",
        "sso:ListInstances"
      ],
      "Resource": "*"
```

```
    },
    {
      "Sid": "DependentIdentityStoreActions",
      "Effect": "Allow",
      "Action": [
        "identitystore:DescribeGroup",
        "identitystore:DescribeUser",
        "identitystore:ListGroupMemberships",
        "sso-directory:DescribeGroup",
        "sso-directory:DescribeGroups",
        "sso-directory:DescribeUser",
        "sso-directory:DescribeUsers",
        "sso-directory:ListGroupsForUser",
        "sso-directory:ListMembersInGroup",
        "sso-directory:SearchGroups",
        "sso-directory:SearchUsers"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DependentOrganizationsActions",
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeAccount",
        "organizations:DescribeOrganization",
        "organizations:ListAccounts",
        "organizations:ListAccountsForParent",
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:ListDelegatedAdministrators",
        "organizations:ListOrganizationalUnitsForParent",
        "organizations:ListRoots"
      ],
      "Resource": "*"
    }
  ]
}
```

To assign IAM roles to workforce users and groups from a delegated administration account without administering account access manager settings you need the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "AAMManageEntitlementsOnly",
      "Effect": "Allow",
      "Action": [
        "account-access:CreateEntitlement",
        "account-access:DeleteEntitlement",
        "account-access:GetApplication",
        "account-access:GetEntitlement",
        "account-access:ListEntitlements"
      ],
      "Resource": "arn:aws:account-access:*:111122223333:application/*"
    },
    {
      "Sid": "AAMListApplications",
      "Effect": "Allow",
      "Action": "account-access:ListApplications",
      "Resource": "*"
    },
    {
      "Sid": "DependentIdentityCenterActions",
      "Effect": "Allow",
      "Action": "sso:ListInstances",
      "Resource": "*"
    },
    {
      "Sid": "DependentIdentityStoreActions",
      "Effect": "Allow",
      "Action": [
        "identitystore:DescribeGroup",
        "identitystore:DescribeUser",
        "identitystore:ListGroupMemberships",
        "sso-directory:DescribeGroup",
        "sso-directory:DescribeGroups",
        "sso-directory:DescribeUser",
        "sso-directory:DescribeUsers",
        "sso-directory:ListGroupsForUser",
        "sso-directory:ListMembersInGroup",
        "sso-directory:SearchGroups",
        "sso-directory:SearchUsers"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DependentOrganizationsActions",
```

```
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeAccount",
        "organizations:DescribeOrganization",
        "organizations:ListAccounts",
        "organizations:ListAccountsForParent",
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:ListDelegatedAdministrators",
        "organizations:ListOrganizationalUnitsForParent",
        "organizations:ListRoots"
      ],
      "Resource": "*"
    }
  ]
}
```

Data protection in account access manager

This section complements the information provided in Data protection in AWS Identity and Access Management with additional details specific to account access manager.

Use of IAM Identity Center user and group data

Account access manager uses user and group data from IAM Identity Center to manage account assignments. However, account access manager stores and logs only AWS-generated identifiers (user ID and group ID) — it does not store or log user names, email addresses, or other personal attributes.

Logging and monitoring in account access manager

Account access manager uses its own API namespace (`account-access`) and logs API calls through CloudTrail.

For more information about CloudTrail, see the AWS CloudTrail User Guide.

Quotas and limits in account access manager

The following tables describe quotas within account access manager. Quota increase requests must come from a management or delegated administrator account. To increase a quota, see Requesting a quota increase.

Group assignment quota

Resource	Default quota	Can be increased
Maximum number of groups assigned to an IAM role	20	Yes

Throttle limits

Resource	Default quota	Can be increased
Account access API	The account access API has a collective throttle limit of 20 transactions per second (TPS). For read APIs, you can open a support case to request a limit increase. The write API operations have a limit of 15 outstanding asynchron ous calls. This limit cannot be increased.	Yes

Code examples for IAM using AWS SDKs

The following code examples show how to use IAM with an AWS software development kit (SDK).

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Code examples for IAM using AWS SDKs
  - Basic examples for IAM using AWS SDKs
    - Hello IAM
    - Learn the basics of IAM with an AWS SDK
    - Actions for IAM using AWS SDKs
      - Use AddClientIdToOpenIdConnectProvider with a CLI
      - Use AddRoleToInstanceProfile with a CLI
      - Use AddUserToGroup with a CLI
      - Use AttachGroupPolicy with a CLI
      - Use AttachRolePolicy with an AWS SDK or CLI
      - Use AttachUserPolicy with an AWS SDK or CLI
      - Use ChangePassword with a CLI
      - Use CreateAccessKey with an AWS SDK or CLI
      - Use CreateAccountAlias with an AWS SDK or CLI
      - Use CreateGroup with an AWS SDK or CLI
      - Use CreateInstanceProfile with an AWS SDK or CLI
      - Use CreateLoginProfile with a CLI
      - Use CreateOpenIdConnectProvider with a CLI
      - Use CreatePolicy with an AWS SDK or CLI
      - Use CreatePolicyVersion with an AWS SDK or CLI
      - Use CreateRole with an AWS SDK or CLI
      - Use CreateSAMLProvider with an AWS SDK or CLI
      - Use CreateServiceLinkedRole with an AWS SDK or CLI

- Use CreateUser with an AWS SDK or CLI
- Use CreateVirtualMfaDevice with a CLI
- Use DeactivateMfaDevice with a CLI
- Use DeleteAccessKey with an AWS SDK or CLI
- Use DeleteAccountAlias with an AWS SDK or CLI
- Use DeleteAccountPasswordPolicy with a CLI
- Use DeleteGroup with an AWS SDK or CLI
- Use DeleteGroupPolicy with a CLI
- Use DeleteInstanceProfile with an AWS SDK or CLI
- Use DeleteLoginProfile with a CLI
- Use DeleteOpenIdConnectProvider with a CLI
- Use DeletePolicy with an AWS SDK or CLI
- Use DeletePolicyVersion with an AWS SDK or CLI
- Use DeleteRole with an AWS SDK or CLI
- Use DeleteRolePermissionsBoundary with a CLI
- Use DeleteRolePolicy with an AWS SDK or CLI
- Use DeleteSAMLProvider with an AWS SDK or CLI
- Use DeleteServerCertificate with an AWS SDK or CLI
- Use DeleteServiceLinkedRole with an AWS SDK or CLI
- Use DeleteSigningCertificate with a CLI
- Use DeleteUser with an AWS SDK or CLI
- Use DeleteUserPermissionsBoundary with a CLI
- Use DeleteUserPolicy with an AWS SDK or CLI
- Use DeleteVirtualMfaDevice with a CLI
- Use DetachGroupPolicy with a CLI
- Use DetachRolePolicy with an AWS SDK or CLI
- Use DetachUserPolicy with an AWS SDK or CLI
- Use EnableMfaDevice with a CLI
- Use GenerateCredentialReport with an AWS SDK or CLI
- Use GenerateServiceLastAccessedDetails with a CLI

- Use GetAccessKeyLastUsed with an AWS SDK or CLI
- Use GetAccountAuthorizationDetails with an AWS SDK or CLI
- Use GetAccountPasswordPolicy with an AWS SDK or CLI
- Use GetAccountSummary with an AWS SDK or CLI
- Use GetContextKeysForCustomPolicy with a CLI
- Use GetContextKeysForPrincipalPolicy with a CLI
- Use GetCredentialReport with an AWS SDK or CLI
- Use GetGroup with a CLI
- Use GetGroupPolicy with a CLI
- Use GetInstanceProfile with a CLI
- Use GetLoginProfile with a CLI
- Use GetOpenIdConnectProvider with a CLI
- Use GetPolicy with an AWS SDK or CLI
- Use GetPolicyVersion with an AWS SDK or CLI
- Use GetRole with an AWS SDK or CLI
- Use GetRolePolicy with a CLI
- Use GetSamlProvider with a CLI
- Use GetServerCertificate with an AWS SDK or CLI
- Use GetServiceLastAccessedDetails with a CLI
- Use GetServiceLastAccessedDetailsWithEntities with a CLI
- Use GetServiceLinkedRoleDeletionStatus with an AWS SDK or CLI
- Use GetUser with an AWS SDK or CLI
- Use GetUserPolicy with a CLI
- Use ListAccessKeys with an AWS SDK or CLI
- Use ListAccountAliases with an AWS SDK or CLI
- Use ListAttachedGroupPolicies with a CLI
- Use ListAttachedRolePolicies with an AWS SDK or CLI
- Use ListAttachedUserPolicies with a CLI
- Use ListEntitiesForPolicy with a CLI
- Use ListGroupPolicies with a CLI

- Use ListGroups with an AWS SDK or CLI
- Use ListGroupsForUser with a CLI
- Use ListInstanceProfiles with a CLI
- Use ListInstanceProfilesForRole with a CLI
- Use ListMfaDevices with a CLI
- Use ListOpenIdConnectProviders with a CLI
- Use ListPolicies with an AWS SDK or CLI
- Use ListPolicyVersions with an AWS SDK or CLI
- Use ListRolePolicies with an AWS SDK or CLI
- Use ListRoleTags with a CLI
- Use ListRoles with an AWS SDK or CLI
- Use ListSAMLProviders with an AWS SDK or CLI
- Use ListServerCertificates with an AWS SDK or CLI
- Use ListSigningCertificates with a CLI
- Use ListUserPolicies with an AWS SDK or CLI
- Use ListUserTags with a CLI
- Use ListUsers with an AWS SDK or CLI
- Use ListVirtualMfaDevices with a CLI
- Use PutGroupPolicy with a CLI
- Use PutRolePermissionsBoundary with a CLI
- Use PutRolePolicy with an AWS SDK or CLI
- Use PutUserPermissionsBoundary with a CLI
- Use PutUserPolicy with an AWS SDK or CLI
- Use RemoveClientIdFromOpenIdConnectProvider with a CLI
- Use RemoveRoleFromInstanceProfile with a CLI
- Use RemoveUserFromGroup with a CLI
- Use ResyncMfaDevice with a CLI
- Use SetDefaultPolicyVersion with an AWS SDK or CLI
- Use TagRole with a CLI
- Use TagUser with a CLI

    - Use UntagRole with a CLI
    - Use UntagUser with a CLI
    - Use UpdateAccessKey with an AWS SDK or CLI
    - Use UpdateAccountPasswordPolicy with a CLI
    - Use UpdateAssumeRolePolicy with a CLI
    - Use UpdateGroup with a CLI
    - Use UpdateLoginProfile with a CLI
    - Use UpdateOpenIdConnectProviderThumbprint with a CLI
    - Use UpdateRole with a CLI
    - Use UpdateRoleDescription with a CLI
    - Use UpdateSamlProvider with a CLI
    - Use UpdateServerCertificate with an AWS SDK or CLI
    - Use UpdateSigningCertificate with a CLI
    - Use UpdateUser with an AWS SDK or CLI
    - Use UploadServerCertificate with an AWS SDK or CLI
    - Use UploadSigningCertificate with a CLI
- Scenarios for IAM using AWS SDKs
    - Build and manage a resilient service using an AWS SDK
    - Configure container service connectivity
    - Create a container task for the serverless launch type
    - Create a rest API with function proxy integration
    - Create read-only and read-write IAM users using an AWS SDK
    - Creating a container service for virtual machine instances
    - Creating a managed monitoring workspace
    - Creating a monitoring dashboard with function name as a variable
    - Creating your first serverless function
    - Get started with serverless data warehouses using the CLI
    - Getting started with configuration management
    - Getting started with internet of things device protection
    - Getting started with machine learning feature stores

- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with provisioned data warehouse clusters
- Getting started with workflow orchestration
- Manage IAM access keys using an AWS SDK
- Manage IAM policies using an AWS SDK
- Manage IAM roles using an AWS SDK
- Manage your IAM account using an AWS SDK
- Moving hardcoded secrets to secure secret storage
- Allows the AWS Compute Optimizer Automation feature to apply recommended actions
- Policy to enable Automation across your organization
- Policy to enable Automation for your account
- Policy to grant full access to Compute Optimizer Automation for a management account of an organization
- Policy to grant full access to Compute Optimizer Automation for standalone AWS accounts
- Policy to grant read-only access to Compute Optimizer Automation for a management account of an organization
- Policy to grant read-only access to Compute Optimizer Automation for standalone AWS accounts
- Policy to grants service-linked role permissions for Compute Optimization Automation
- Roll back an IAM policy version using an AWS SDK
- Run CPU stress tests on virtual machine instances using fault injection
- Set up Attribute-Based Access Control for DynamoDB using AWS Command Line Interface v2
- Setting up systems management
- Using property variables in monitoring dashboards to monitor multiple serverless functions
- Work with DynamoDB Streams and Time-to-Live using AWS Command Line Interface v2
- Work with the IAM Policy Builder API using an AWS SDK

- Basic examples for AWS STS using AWS SDKs
  - Actions for AWS STS using AWS SDKs
    - Use AssumeRole with an AWS SDK or CLI
    - Use AssumeRoleWithWebIdentity with a CLI
    - Use DecodeAuthorizationMessage with a CLI
    - Use GetFederationToken with a CLI
    - Use GetSessionToken with an AWS SDK or CLI
- Scenarios for AWS STS using AWS SDKs
  - Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK
  - Configure container service connectivity
  - Construct a URL with AWS STS for federated users using an AWS SDK
  - Create a container task for the serverless launch type
  - Create a rest API with function proxy integration
  - Creating a container service for virtual machine instances
  - Creating a managed monitoring workspace
  - Creating a monitoring dashboard with function name as a variable
  - Get a session token that requires an MFA token with AWS STS using an AWS SDK
  - Getting started with configuration management
  - Getting started with container registries
  - Getting started with internet of things messaging
  - Getting started with machine learning feature stores
  - Getting started with managed kubernetes clusters
  - Getting started with managed streaming
  - Getting started with push notifications
  - Getting started with search and analytics engines
  - Getting started with web application firewalls
  - Moving hardcoded secrets to secure secret storage
  - Run CPU stress tests on virtual machine instances using fault injection
  - Setting up systems management

Code examples for IAM using AWS SDKs

The following code examples show how to use IAM with an AWS software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

IAM

- Basic examples for IAM using AWS SDKs
  - Hello IAM
  - Learn the basics of IAM with an AWS SDK
  - Actions for IAM using AWS SDKs
    - Use AddClientIdToOpenIdConnectProvider with a CLI
    - Use AddRoleToInstanceProfile with a CLI
    - Use AddUserToGroup with a CLI
    - Use AttachGroupPolicy with a CLI
    - Use AttachRolePolicy with an AWS SDK or CLI
    - Use AttachUserPolicy with an AWS SDK or CLI
    - Use ChangePassword with a CLI
    - Use CreateAccessKey with an AWS SDK or CLI
    - Use CreateAccountAlias with an AWS SDK or CLI
    - Use CreateGroup with an AWS SDK or CLI
    - Use CreateInstanceProfile with an AWS SDK or CLI
    - Use CreateLoginProfile with a CLI
    - Use CreateOpenIdConnectProvider with a CLI
    - Use CreatePolicy with an AWS SDK or CLI

- Use CreatePolicyVersion with an AWS SDK or CLI
- Use CreateRole with an AWS SDK or CLI
- Use CreateSAMLProvider with an AWS SDK or CLI
- Use CreateServiceLinkedRole with an AWS SDK or CLI
- Use CreateUser with an AWS SDK or CLI
- Use CreateVirtualMfaDevice with a CLI
- Use DeactivateMfaDevice with a CLI
- Use DeleteAccessKey with an AWS SDK or CLI
- Use DeleteAccountAlias with an AWS SDK or CLI
- Use DeleteAccountPasswordPolicy with a CLI
- Use DeleteGroup with an AWS SDK or CLI
- Use DeleteGroupPolicy with a CLI
- Use DeleteInstanceProfile with an AWS SDK or CLI
- Use DeleteLoginProfile with a CLI
- Use DeleteOpenIdConnectProvider with a CLI
- Use DeletePolicy with an AWS SDK or CLI
- Use DeletePolicyVersion with an AWS SDK or CLI
- Use DeleteRole with an AWS SDK or CLI
- Use DeleteRolePermissionsBoundary with a CLI
- Use DeleteRolePolicy with an AWS SDK or CLI
- Use DeleteSAMLProvider with an AWS SDK or CLI
- Use DeleteServerCertificate with an AWS SDK or CLI
- Use DeleteServiceLinkedRole with an AWS SDK or CLI
- Use DeleteSigningCertificate with a CLI
- Use DeleteUser with an AWS SDK or CLI
- Use DeleteUserPermissionsBoundary with a CLI
- Use DeleteUserPolicy with an AWS SDK or CLI
- Use DeleteVirtualMfaDevice with a CLI
- Use DetachGroupPolicy with a CLI
- Use DetachRolePolicy with an AWS SDK or CLI

- Use DetachUserPolicy with an AWS SDK or CLI
- Use EnableMfaDevice with a CLI
- Use GenerateCredentialReport with an AWS SDK or CLI
- Use GenerateServiceLastAccessedDetails with a CLI
- Use GetAccessKeyLastUsed with an AWS SDK or CLI
- Use GetAccountAuthorizationDetails with an AWS SDK or CLI
- Use GetAccountPasswordPolicy with an AWS SDK or CLI
- Use GetAccountSummary with an AWS SDK or CLI
- Use GetContextKeysForCustomPolicy with a CLI
- Use GetContextKeysForPrincipalPolicy with a CLI
- Use GetCredentialReport with an AWS SDK or CLI
- Use GetGroup with a CLI
- Use GetGroupPolicy with a CLI
- Use GetInstanceProfile with a CLI
- Use GetLoginProfile with a CLI
- Use GetOpenIdConnectProvider with a CLI
- Use GetPolicy with an AWS SDK or CLI
- Use GetPolicyVersion with an AWS SDK or CLI
- Use GetRole with an AWS SDK or CLI
- Use GetRolePolicy with a CLI
- Use GetSamlProvider with a CLI
- Use GetServerCertificate with an AWS SDK or CLI
- Use GetServiceLastAccessedDetails with a CLI
- Use GetServiceLastAccessedDetailsWithEntities with a CLI
- Use GetServiceLinkedRoleDeletionStatus with an AWS SDK or CLI
- Use GetUser with an AWS SDK or CLI
- Use GetUserPolicy with a CLI
- Use ListAccessKeys with an AWS SDK or CLI
- Use ListAccountAliases with an AWS SDK or CLI
- Use ListAttachedGroupPolicies with a CLI

- Use ListAttachedRolePolicies with an AWS SDK or CLI
- Use ListAttachedUserPolicies with a CLI
- Use ListEntitiesForPolicy with a CLI
- Use ListGroupPolicies with a CLI
- Use ListGroups with an AWS SDK or CLI
- Use ListGroupsForUser with a CLI
- Use ListInstanceProfiles with a CLI
- Use ListInstanceProfilesForRole with a CLI
- Use ListMfaDevices with a CLI
- Use ListOpenIdConnectProviders with a CLI
- Use ListPolicies with an AWS SDK or CLI
- Use ListPolicyVersions with an AWS SDK or CLI
- Use ListRolePolicies with an AWS SDK or CLI
- Use ListRoleTags with a CLI
- Use ListRoles with an AWS SDK or CLI
- Use ListSAMLProviders with an AWS SDK or CLI
- Use ListServerCertificates with an AWS SDK or CLI
- Use ListSigningCertificates with a CLI
- Use ListUserPolicies with an AWS SDK or CLI
- Use ListUserTags with a CLI
- Use ListUsers with an AWS SDK or CLI
- Use ListVirtualMfaDevices with a CLI
- Use PutGroupPolicy with a CLI
- Use PutRolePermissionsBoundary with a CLI
- Use PutRolePolicy with an AWS SDK or CLI
- Use PutUserPermissionsBoundary with a CLI
- Use PutUserPolicy with an AWS SDK or CLI
- Use RemoveClientIdFromOpenIdConnectProvider with a CLI
- Use RemoveRoleFromInstanceProfile with a CLI
- Use RemoveUserFromGroup with a CLI

- Use ResyncMfaDevice with a CLI
- Use SetDefaultPolicyVersion with an AWS SDK or CLI
- Use TagRole with a CLI
- Use TagUser with a CLI
- Use UntagRole with a CLI
- Use UntagUser with a CLI
- Use UpdateAccessKey with an AWS SDK or CLI
- Use UpdateAccountPasswordPolicy with a CLI
- Use UpdateAssumeRolePolicy with a CLI
- Use UpdateGroup with a CLI
- Use UpdateLoginProfile with a CLI
- Use UpdateOpenIdConnectProviderThumbprint with a CLI
- Use UpdateRole with a CLI
- Use UpdateRoleDescription with a CLI
- Use UpdateSamlProvider with a CLI
- Use UpdateServerCertificate with an AWS SDK or CLI
- Use UpdateSigningCertificate with a CLI
- Use UpdateUser with an AWS SDK or CLI
- Use UploadServerCertificate with an AWS SDK or CLI
- Use UploadSigningCertificate with a CLI

- Scenarios for IAM using AWS SDKs
  - Build and manage a resilient service using an AWS SDK
  - Configure container service connectivity
  - Create a container task for the serverless launch type
  - Create a rest API with function proxy integration
  - Create read-only and read-write IAM users using an AWS SDK
  - Creating a container service for virtual machine instances
  - Creating a managed monitoring workspace
  - Creating a monitoring dashboard with function name as a variable
  - Creating your first serverless function

- Get started with serverless data warehouses using the CLI
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with provisioned data warehouse clusters
- Getting started with workflow orchestration
- Manage IAM access keys using an AWS SDK
- Manage IAM policies using an AWS SDK
- Manage IAM roles using an AWS SDK
- Manage your IAM account using an AWS SDK
- Moving hardcoded secrets to secure secret storage
- Allows the AWS Compute Optimizer Automation feature to apply recommended actions
- Policy to enable Automation across your organization
- Policy to enable Automation for your account
- Policy to grant full access to Compute Optimizer Automation for a management account of an organization
- Policy to grant full access to Compute Optimizer Automation for standalone AWS accounts
- Policy to grant read-only access to Compute Optimizer Automation for a management account of an organization
- Policy to grant read-only access to Compute Optimizer Automation for standalone AWS accounts
- Policy to grants service-linked role permissions for Compute Optimization Automation
- Roll back an IAM policy version using an AWS SDK
- Run CPU stress tests on virtual machine instances using fault injection
- Set up Attribute-Based Access Control for DynamoDB using AWS Command Line Interface v2
- Setting up systems management
- Using property variables in monitoring dashboards to monitor multiple serverless functions
- Work with DynamoDB Streams and Time-to-Live using AWS Command Line Interface v2
- Work with the IAM Policy Builder API using an AWS SDK

Basic examples for IAM using AWS SDKs

The following code examples show how to use the basics of AWS Identity and Access Management with AWS SDKs.

Examples

- Hello IAM
- Learn the basics of IAM with an AWS SDK
- Actions for IAM using AWS SDKs
  - Use AddClientIdToOpenIdConnectProvider with a CLI
  - Use AddRoleToInstanceProfile with a CLI
  - Use AddUserToGroup with a CLI
  - Use AttachGroupPolicy with a CLI
  - Use AttachRolePolicy with an AWS SDK or CLI
  - Use AttachUserPolicy with an AWS SDK or CLI
  - Use ChangePassword with a CLI
  - Use CreateAccessKey with an AWS SDK or CLI
  - Use CreateAccountAlias with an AWS SDK or CLI
  - Use CreateGroup with an AWS SDK or CLI
  - Use CreateInstanceProfile with an AWS SDK or CLI
  - Use CreateLoginProfile with a CLI
  - Use CreateOpenIdConnectProvider with a CLI
  - Use CreatePolicy with an AWS SDK or CLI
  - Use CreatePolicyVersion with an AWS SDK or CLI
  - Use CreateRole with an AWS SDK or CLI
  - Use CreateSAMLProvider with an AWS SDK or CLI
  - Use CreateServiceLinkedRole with an AWS SDK or CLI
  - Use CreateUser with an AWS SDK or CLI
  - Use CreateVirtualMfaDevice with a CLI
  - Use DeactivateMfaDevice with a CLI
  - Use DeleteAccessKey with an AWS SDK or CLI
  - Use DeleteAccountAlias with an AWS SDK or CLI

- Use DeleteAccountPasswordPolicy with a CLI
- Use DeleteGroup with an AWS SDK or CLI
- Use DeleteGroupPolicy with a CLI
- Use DeleteInstanceProfile with an AWS SDK or CLI
- Use DeleteLoginProfile with a CLI
- Use DeleteOpenIdConnectProvider with a CLI
- Use DeletePolicy with an AWS SDK or CLI
- Use DeletePolicyVersion with an AWS SDK or CLI
- Use DeleteRole with an AWS SDK or CLI
- Use DeleteRolePermissionsBoundary with a CLI
- Use DeleteRolePolicy with an AWS SDK or CLI
- Use DeleteSAMLProvider with an AWS SDK or CLI
- Use DeleteServerCertificate with an AWS SDK or CLI
- Use DeleteServiceLinkedRole with an AWS SDK or CLI
- Use DeleteSigningCertificate with a CLI
- Use DeleteUser with an AWS SDK or CLI
- Use DeleteUserPermissionsBoundary with a CLI
- Use DeleteUserPolicy with an AWS SDK or CLI
- Use DeleteVirtualMfaDevice with a CLI
- Use DetachGroupPolicy with a CLI
- Use DetachRolePolicy with an AWS SDK or CLI
- Use DetachUserPolicy with an AWS SDK or CLI
- Use EnableMfaDevice with a CLI
- Use GenerateCredentialReport with an AWS SDK or CLI
- Use GenerateServiceLastAccessedDetails with a CLI
- Use GetAccessKeyLastUsed with an AWS SDK or CLI
- Use GetAccountAuthorizationDetails with an AWS SDK or CLI
- Use GetAccountPasswordPolicy with an AWS SDK or CLI
- Use GetAccountSummary with an AWS SDK or CLI

- Use GetContextKeysForCustomPolicy with a CLI

- Use GetContextKeysForPrincipalPolicy with a CLI
- Use GetCredentialReport with an AWS SDK or CLI
- Use GetGroup with a CLI
- Use GetGroupPolicy with a CLI
- Use GetInstanceProfile with a CLI
- Use GetLoginProfile with a CLI
- Use GetOpenIdConnectProvider with a CLI
- Use GetPolicy with an AWS SDK or CLI
- Use GetPolicyVersion with an AWS SDK or CLI
- Use GetRole with an AWS SDK or CLI
- Use GetRolePolicy with a CLI
- Use GetSamlProvider with a CLI
- Use GetServerCertificate with an AWS SDK or CLI
- Use GetServiceLastAccessedDetails with a CLI
- Use GetServiceLastAccessedDetailsWithEntities with a CLI
- Use GetServiceLinkedRoleDeletionStatus with an AWS SDK or CLI
- Use GetUser with an AWS SDK or CLI
- Use GetUserPolicy with a CLI
- Use ListAccessKeys with an AWS SDK or CLI
- Use ListAccountAliases with an AWS SDK or CLI
- Use ListAttachedGroupPolicies with a CLI
- Use ListAttachedRolePolicies with an AWS SDK or CLI
- Use ListAttachedUserPolicies with a CLI
- Use ListEntitiesForPolicy with a CLI
- Use ListGroupPolicies with a CLI
- Use ListGroups with an AWS SDK or CLI
- Use ListGroupsForUser with a CLI
- Use ListInstanceProfiles with a CLI
- Use ListInstanceProfilesForRole with a CLI

- Use ListMfaDevices with a CLI

- Use ListOpenIdConnectProviders with a CLI
- Use ListPolicies with an AWS SDK or CLI
- Use ListPolicyVersions with an AWS SDK or CLI
- Use ListRolePolicies with an AWS SDK or CLI
- Use ListRoleTags with a CLI
- Use ListRoles with an AWS SDK or CLI
- Use ListSAMLProviders with an AWS SDK or CLI
- Use ListServerCertificates with an AWS SDK or CLI
- Use ListSigningCertificates with a CLI
- Use ListUserPolicies with an AWS SDK or CLI
- Use ListUserTags with a CLI
- Use ListUsers with an AWS SDK or CLI
- Use ListVirtualMfaDevices with a CLI
- Use PutGroupPolicy with a CLI
- Use PutRolePermissionsBoundary with a CLI
- Use PutRolePolicy with an AWS SDK or CLI
- Use PutUserPermissionsBoundary with a CLI
- Use PutUserPolicy with an AWS SDK or CLI
- Use RemoveClientIdFromOpenIdConnectProvider with a CLI
- Use RemoveRoleFromInstanceProfile with a CLI
- Use RemoveUserFromGroup with a CLI
- Use ResyncMfaDevice with a CLI
- Use SetDefaultPolicyVersion with an AWS SDK or CLI
- Use TagRole with a CLI
- Use TagUser with a CLI
- Use UntagRole with a CLI
- Use UntagUser with a CLI
- Use UpdateAccessKey with an AWS SDK or CLI
- Use UpdateAccountPasswordPolicy with a CLI
- Use UpdateAssumeRolePolicy with a CLI

- Use UpdateGroup with a CLI
- Use UpdateLoginProfile with a CLI
- Use UpdateOpenIdConnectProviderThumbprint with a CLI
- Use UpdateRole with a CLI
- Use UpdateRoleDescription with a CLI
- Use UpdateSamlProvider with a CLI
- Use UpdateServerCertificate with an AWS SDK or CLI
- Use UpdateSigningCertificate with a CLI
- Use UpdateUser with an AWS SDK or CLI
- Use UploadServerCertificate with an AWS SDK or CLI
- Use UploadSigningCertificate with a CLI

Hello IAM

The following code examples show how to get started using IAM.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace IAMActions;

public class HelloIAM
{
    static async Task Main(string[] args)
    {
        // Getting started with AWS Identity and Access Management (IAM). List
        // the policies for the account.
        var iamClient = new AmazonIdentityManagementServiceClient();
```

```
        var listPoliciesPaginator = iamClient.Paginators.ListPolicies(new
 ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        Console.WriteLine("Here are the policies defined for your account:\n");
        policies.ForEach(policy =>
        {
            Console.WriteLine($"Created:
 {policy.CreateDate}\t{policy.PolicyName}\t{policy.Description}");
        });
    }
}
```

- For API details, see ListPolicies in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS iam)

# Set this project's name.
project("hello_iam")
```

```
# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_iam.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code for the iam.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/iam/IAMClient.h>
#include <aws/iam/model/ListPoliciesRequest.h>
#include <iostream>
```

```
#include <iomanip>

/*
 *  A "Hello IAM" starter application which initializes an AWS Identity and
 Access Management (IAM) client
 *  and lists the IAM policies.
 *
 *  main function
 *
 *  Usage: 'hello_iam'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        const Aws::String DATE_FORMAT("%Y-%m-%d");
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::IAM::IAMClient iamClient(clientConfig);
        Aws::IAM::Model::ListPoliciesRequest request;

        bool done = false;
        bool header = false;
        while (!done) {
            auto outcome = iamClient.ListPolicies(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Failed to list iam policies: " <<
                          outcome.GetError().GetMessage() << std::endl;
                result = 1;
                break;
            }

            if (!header) {
                std::cout << std::left << std::setw(55) << "Name" <<
                          std::setw(30) << "ID" << std::setw(80) << "Arn" <<
                          std::setw(64) << "Description" << std::setw(12) <<
                          "CreateDate" << std::endl;
```

```
                header = true;
            }

            const auto &policies = outcome.GetResult().GetPolicies();
            for (const auto &policy: policies) {
                std::cout << std::left << std::setw(55) <<
                          policy.GetPolicyName() << std::setw(30) <<
                          policy.GetPolicyId() << std::setw(80) <<
 policy.GetArn() <<
                          std::setw(64) << policy.GetDescription() <<
 std::setw(12) <<
                          policy.GetCreateDate().ToGmtString(DATE_FORMAT.c_str())
 <<
                          std::endl;
            }

            if (outcome.GetResult().GetIsTruncated()) {
                request.SetMarker(outcome.GetResult().GetMarker());
            } else {
                done = true;
            }
        }
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- For API details, see ListPolicies in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/iam"
)

// main uses the AWS SDK for Go (v2) to create an AWS Identity and Access
 Management (IAM)
// client and list up to 10 policies in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 iamClient := iam.NewFromConfig(sdkConfig)
 const maxPols = 10
 fmt.Printf("Let's list up to %v policies for your account.\n", maxPols)
 result, err := iamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPols),
 })
 if err != nil {
  fmt.Printf("Couldn't list policies for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Policies) == 0 {
  fmt.Println("You don't have any policies!")
 } else {
  for _, policy := range result.Policies {
   fmt.Printf("\t%v\n", *policy.PolicyName)
  }
 }
```

```
}
```

- For API details, see ListPolicies in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.ListPoliciesResponse;
import software.amazon.awssdk.services.iam.model.Policy;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloIAM {
    public static void main(String[] args) {
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        listPolicies(iam);
    }

    public static void listPolicies(IamClient iam) {
```

```
        ListPoliciesResponse response = iam.listPolicies();
        List<Policy> polList = response.policies();
        polList.forEach(policy -> {
            System.out.println("Policy Name: " + policy.policyName());
        });
    }
}
```

- For API details, see ListPolicies in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
 async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      for (const policy of page.Policies) {
```

```
        console.log(`${policy.PolicyName}`);
        policyCount++;
      }
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- For API details, see ListPolicies in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import boto3


def main():
    """
    Lists the managed policies in your AWS account using the AWS SDK for Python
 (Boto3).
    """
    iam = boto3.client("iam")

    try:
        # Get a paginator for the list_policies operation
        paginator = iam.get_paginator("list_policies")

        # Iterate through the pages of results
        for page in paginator.paginate(Scope="All", OnlyAttached=False):
            for policy in page["Policies"]:
                print(f"Policy name: {policy['PolicyName']}")
                print(f"  Policy ARN: {policy['Arn']}")
    except boto3.exceptions.BotoCoreError as e:
```

```
        print(f"Encountered an error while listing policies: {e}")


if __name__ == "__main__":
    main()
```

- For API details, see ListPolicies in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-iam'
require 'logger'

# IAMManager is a class responsible for managing IAM operations
# such as listing all IAM policies in the current AWS account.
class IAMManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all IAM policies in the current AWS account.
  def list_policies
    @logger.info('Here are the IAM policies in your account:')

    paginator = @client.list_policies
    policies = []

    paginator.each_page do |page|
      policies.concat(page.policies)
    end
```

```
    if policies.empty?
      @logger.info("You don't have any IAM policies.")
    else
      policies.each do |policy|
        @logger.info("- #{policy.policy_name}")
      end
    end
  end
end

if $PROGRAM_NAME == __FILE__
  iam_client = Aws::IAM::Client.new
  manager = IAMManager.new(iam_client)
  manager.list_policies
end
```

- For API details, see ListPolicies in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

From src/bin/hello.rs.

```
use aws_sdk_iam::error::SdkError;
use aws_sdk_iam::operation::list_policies::ListPoliciesError;
use clap::Parser;

const PATH_PREFIX_HELP: &str = "The path prefix for filtering the results.";

#[derive(Debug, clap::Parser)]
#[command(about)]
```

```
struct HelloScenarioArgs {
    #[arg(long, default_value="/", help=PATH_PREFIX_HELP)]
    pub path_prefix: String,
}

#[tokio::main]
async fn main() -> Result<(), Box<SdkError<ListPoliciesError>>> {
    let sdk_config = aws_config::load_from_env().await;
    let client = aws_sdk_iam::Client::new(&sdk_config);

    let args = HelloScenarioArgs::parse();

    iam_service::list_policies(client, args.path_prefix).await?;

    Ok(())
}
```

From src/iam-service-lib.rs.

```
pub async fn list_policies(
    client: iamClient,
    path_prefix: String,
) -> Result<Vec<String>, SdkError<ListPoliciesError>> {
    let list_policies = client
        .list_policies()
        .path_prefix(path_prefix)
        .scope(PolicyScopeType::Local)
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await?;

    let policy_names = list_policies
        .into_iter()
        .map(|p| {
            let name = p
                .policy_name
                .unwrap_or_else(|| "Missing Policy Name".to_string());
            println!("{}", name);
            name
        })
```

```
        .collect();

    Ok(policy_names)
}
```

- For API details, see ListPolicies in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Learn the basics of IAM with an AWS SDK

The following code examples show how to create a user and assume a role.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;


namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
 AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
 CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;

    }


    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</
param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
 policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new
 CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }
```

```
    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
 rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }


    /// <summary>
    /// Create an IAM service-linked role.
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName,
 string description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }


    /// <summary>
```

```
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }


    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new
DeletePolicyRequest { PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
    }


    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
 { RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role policy.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
 policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
 DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
```

```
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM
policy.</param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
 GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }


    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</
param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
 { PolicyArn = policyArn });
        return response.Policy;
    }


    /// <summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
            RoleName = roleName,
        });

        return response.Role;
    }
```

```
    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest
{ UserName = userName });
        return response.User;
    }


    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>>
ListAttachedRolePoliciesAsync(string roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new
ListAttachedRolePoliciesRequest { RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }


    /// <summary>
    /// List IAM groups.
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();
```

```
        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }


    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
 ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }


    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</
param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator =
 _IAMService.Paginators.ListRolePolicies(new ListRolePoliciesRequest { RoleName =
 roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
```

```
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }


    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }


    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }


    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
```

```
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string
 roleName, string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM
 policy.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> PutUserPolicyAsync(string userName, string
 policyName, string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Wait for a new access key to be ready to use.
    /// </summary>
    /// <param name="accessKeyId">The Id of the access key.</param>
    /// <returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId =
 accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}
```

```
using Microsoft.Extensions.Configuration;

namespace IAMBasics;

public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>()
            .AddTransient<IAMWrapper>()
            .AddTransient<UIWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<IAMBasics>();


        IConfiguration configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // Values needed for user, role, and policies.
        string userName = configuration["UserName"]!;
        string s3PolicyName = configuration["S3PolicyName"]!;
        string roleName = configuration["RoleName"]!;
```

```
        var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        uiWrapper.DisplayBasicsOverview();
        uiWrapper.PressEnter();

        // First create a user. By default, the new user has
        // no permissions.
        uiWrapper.DisplayTitle("Create User");
        Console.WriteLine($"Creating a new user with user name: {userName}.");
        var user = await iamWrapper.CreateUserAsync(userName);
        var userArn = user.Arn;

        Console.WriteLine($"Successfully created user: {userName} with ARN:
{userArn}.");
        uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for
use.");

        // Define a role policy document that allows the new user
        // to assume the role.
        string assumeRolePolicyDocument = "{" +
          "\"Version\": \"2012-10-17\"," +
          "\"Statement\": [{" +
              "\"Effect\": \"Allow\"," +
              "\"Principal\": {" +
              $" \"AWS\": \"{userArn}\"" +
              "}," +
              "\"Action\": \"sts:AssumeRole\"" +
          "}]" +
        "}";

        // Permissions to list all buckets.
        string policyDocument = "{" +
            "\"Version\": \"2012-10-17\"," +
            " \"Statement\" : [{" +
                " \"Action\" : [\"s3:ListAllMyBuckets\"]," +
                " \"Effect\" : \"Allow\"," +
                " \"Resource\" : \"*\"" +
            "}]" +
        "}";

        // Create an AccessKey for the user.
        uiWrapper.DisplayTitle("Create access key");
```

```
        Console.WriteLine("Now let's create an access key for the new user.");
        var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

        var accessKeyId = accessKey.AccessKeyId;
        var secretAccessKey = accessKey.SecretAccessKey;

        Console.WriteLine($"We have created the access key with Access key id:
{accessKeyId}.");

        Console.WriteLine("Now let's wait until the IAM access key is ready to
use.");
        var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

        // Now try listing the Amazon Simple Storage Service (Amazon S3)
        // buckets. This should fail at this point because the user doesn't
        // have permissions to perform this task.
        uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
        Console.WriteLine("Now let's try to display a list of the user's Amazon
S3 buckets.");
        var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
        var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

        var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
        var buckets = await s3Wrapper.ListMyBucketsAsync();

        Console.WriteLine(buckets is null
            ? "As expected, the call to list the buckets has returned a null
list."
            : "Something went wrong. This shouldn't have worked.");

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create IAM role");
        Console.WriteLine($"Creating the role: {roleName}");

        // Creating an IAM role to allow listing the S3 buckets. A role name
        // is not case sensitive and must be unique to the account for which it
        // is created.
        var roleArn = await iamWrapper.CreateRoleAsync(roleName,
assumeRolePolicyDocument);

        uiWrapper.PressEnter();
```

```
        // Create a policy with permissions to list S3 buckets.
        uiWrapper.DisplayTitle("Create IAM policy");
        Console.WriteLine($"Creating the policy: {s3PolicyName}");
        Console.WriteLine("with permissions to list the Amazon S3 buckets for the
account.");
        var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
policyDocument);

        // Wait 15 seconds for the IAM policy to be available.
        uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

        // Attach the policy to the role you created earlier.
        uiWrapper.DisplayTitle("Attach new IAM policy");
        Console.WriteLine("Now let's attach the policy to the role.");
        await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);

        // Wait 15 seconds for the role to be updated.
        Console.WriteLine();
        uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

        // Use the AWS Security Token Service (AWS STS) to have the user
        // assume the role we created.
        var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

        // Wait for the new credentials to become valid.
        uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");

        var assumedRoleCredentials = await
s3Wrapper.AssumeS3RoleAsync("temporary-session", roleArn);

        // Try again to list the buckets using the client created with
        // the new user's credentials. This time, it should work.
        var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

        s3Wrapper.UpdateClients(s3Client2, stsClient2);

        buckets = await s3Wrapper.ListMyBucketsAsync();

        uiWrapper.DisplayTitle("List Amazon S3 buckets");
        Console.WriteLine("This time we should have buckets to list.");
        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
```

```
            {
                Console.WriteLine($"{bucket.BucketName} created:
 {bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        // Now clean up all the resources used in the example.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Thank you for watching. The IAM Basics demo is
 complete.");
        Console.WriteLine("Please wait while we clean up the resources we
 created.");

        await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);

        await iamWrapper.DeletePolicyAsync(policy.Arn);

        await iamWrapper.DeleteRoleAsync(roleName);

        await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

        await iamWrapper.DeleteUserAsync(userName);

        uiWrapper.PressEnter();

        Console.WriteLine("All done cleaning up our resources. Thank you for your
 patience.");
    }
}


namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
```

```
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</
param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
 roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }


    /// <summary>
    /// Delete an S3 bucket.
    /// </summary>
    /// <param name="bucketName">Name of the S3 bucket to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
```

```
    {
        var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
        return result.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the buckets that are owned by the user's account.
    /// </summary>
    /// <returns>Async Task.</returns>
    public async Task<List<S3Bucket>?> ListMyBucketsAsync()
    {
        try
        {
            // Get the list of buckets accessible by the new user.
            var response = await _s3Service.ListBucketsAsync();

            return response.Buckets;
        }
        catch (AmazonS3Exception ex)
        {
            // Something else went wrong. Display the error message.
            Console.WriteLine($"Error: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>
    /// <returns>A Boolean value indicating whether the action completed
    /// successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
```

```
    /// </summary>
    /// <param name="s3Service">The Amazon S3 client object.</param>
    /// <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
 stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}


namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
 (IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it
 full access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
 creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    /// <summary>
    /// Show information about the IAM Basics scenario.
    /// </summary>
```

```
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries
to list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the
role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title, and another
    /// line of hyphens.
```

```
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser

- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
###############################################################################
# function iam_create_user_assume_role
#
# Scenario to create an IAM user, create an IAM role, and apply the role to the
 user.
#
#     "IAM access" permissions are needed to run this code.
#     "STS assume role" permissions are needed to run this code. (Note: It might
 be necessary to
#           create a custom policy).
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
###############################################################################
function iam_create_user_assume_role() {
  {
    if [ "$IAM_OPERATIONS_SOURCED" != "True" ]; then

      source ./iam_operations.sh
    fi
  }

  echo_repeat "*" 88
  echo "Welcome to the IAM create user and assume role demo."
  echo
```

```
 echo "This demo will create an IAM user, create an IAM role, and apply the role
to the user."
 echo_repeat "*" 88
 echo

 echo -n "Enter a name for a new IAM user: "
 get_input
 user_name=$get_input_result

 local user_arn
 user_arn=$(iam_create_user -u "$user_name")

 # shellcheck disable=SC2181
 if [[ ${?} == 0 ]]; then
   echo "Created demo IAM user named $user_name"
 else
   errecho "$user_arn"
   errecho "The user failed to create. This demo will exit."
   return 1
 fi

 local access_key_response
 access_key_response=$(iam_create_user_access_key -u "$user_name")
 # shellcheck disable=SC2181
 if [[ ${?} != 0 ]]; then
   errecho "The access key failed to create. This demo will exit."
   clean_up "$user_name"
   return 1
 fi

 IFS=$'\t ' read -r -a access_key_values <<<"$access_key_response"
 local key_name=${access_key_values[0]}
 local key_secret=${access_key_values[1]}

 echo "Created access key named $key_name"

 echo "Wait 10 seconds for the user to be ready."
 sleep 10
 echo_repeat "*" 88
 echo

 local iam_role_name
 iam_role_name=$(generate_random_name "test-role")
```

```
 echo "Creating a role named $iam_role_name with user $user_name as the
principal."

 local assume_role_policy_document="{
   \"Version\": \"2012-10-17\",
   \"Statement\": [{
       \"Effect\": \"Allow\",
       \"Principal\": {\"AWS\": \"$user_arn\"},
       \"Action\": \"sts:AssumeRole\"
       }]
   }"

 local role_arn
 role_arn=$(iam_create_role -n "$iam_role_name" -p
"$assume_role_policy_document")

 # shellcheck disable=SC2181
 if [ ${?} == 0 ]; then
   echo "Created IAM role named $iam_role_name"
 else
   errecho "The role failed to create. This demo will exit."
   clean_up "$user_name" "$key_name"
   return 1
 fi

 local policy_name
 policy_name=$(generate_random_name "test-policy")
 local policy_document="{
              \"Version\": \"2012-10-17\",
              \"Statement\": [{
                  \"Effect\": \"Allow\",
                  \"Action\": \"s3:ListAllMyBuckets\",
                  \"Resource\": \"arn:aws:s3:::*\"}]}"

 local policy_arn
 policy_arn=$(iam_create_policy -n "$policy_name" -p "$policy_document")
 # shellcheck disable=SC2181
 if [[ ${?} == 0 ]]; then
   echo "Created  IAM policy named $policy_name"
 else
   errecho "The policy failed to create."
   clean_up "$user_name" "$key_name" "$iam_role_name"
   return 1
 fi
```

```
 if (iam_attach_role_policy -n "$iam_role_name" -p "$policy_arn"); then
   echo "Attached policy $policy_arn to role $iam_role_name"
 else
   errecho "The policy failed to attach."
   clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
   return 1
 fi

 local assume_role_policy_document="{
               \"Version\": \"2012-10-17\",
               \"Statement\": [{
                   \"Effect\": \"Allow\",
                   \"Action\": \"sts:AssumeRole\",
                   \"Resource\": \"$role_arn\"}]}"

 local assume_role_policy_name
 assume_role_policy_name=$(generate_random_name "test-assume-role-")

 # shellcheck disable=SC2181
 local assume_role_policy_arn
 assume_role_policy_arn=$(iam_create_policy -n "$assume_role_policy_name" -p
"$assume_role_policy_document")
 # shellcheck disable=SC2181
 if [ ${?} == 0 ]; then
   echo "Created  IAM policy named $assume_role_policy_name for sts assume role"
 else
   errecho "The policy failed to create."
   clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
"$policy_arn"
   return 1
 fi

 echo "Wait 10 seconds to give AWS time to propagate these new resources and
connections."
 sleep 10
 echo_repeat "*" 88
 echo

 echo "Try to list buckets without the new user assuming the role."
 echo_repeat "*" 88
 echo

 # Set the environment variables for the created user.
```

```
 # bashsupport disable=BP2001
 export AWS_ACCESS_KEY_ID=$key_name
 # bashsupport disable=BP2001
 export AWS_SECRET_ACCESS_KEY=$key_secret

 local buckets
 buckets=$(s3_list_buckets)

 # shellcheck disable=SC2181
 if [ ${?} == 0 ]; then
   local bucket_count
   bucket_count=$(echo "$buckets" | wc -w | xargs)
   echo "There are $bucket_count buckets in the account. This should not have
happened."
 else
   errecho "Because the role with permissions has not been assumed, listing
buckets failed."
 fi

 echo
 echo_repeat "*" 88
 echo "Now assume the role $iam_role_name and list the buckets."
 echo_repeat "*" 88
 echo

 local credentials

 credentials=$(sts_assume_role -r "$role_arn" -n "AssumeRoleDemoSession")
 # shellcheck disable=SC2181
 if [ ${?} == 0 ]; then
   echo "Assumed role $iam_role_name"
 else
   errecho "Failed to assume role."
   export AWS_ACCESS_KEY_ID=""
   export AWS_SECRET_ACCESS_KEY=""
   clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
"$policy_arn" "$assume_role_policy_arn"
   return 1
 fi

 IFS=$'\t ' read -r -a credentials <<<"$credentials"

 export AWS_ACCESS_KEY_ID=${credentials[0]}
 export AWS_SECRET_ACCESS_KEY=${credentials[1]}
```

```
  # bashsupport disable=BP2001
  export AWS_SESSION_TOKEN=${credentials[2]}

  buckets=$(s3_list_buckets)

  # shellcheck disable=SC2181
  if [ ${?} == 0 ]; then
    local bucket_count
    bucket_count=$(echo "$buckets" | wc -w | xargs)
    echo "There are $bucket_count buckets in the account. Listing buckets
 succeeded because of "
    echo "the assumed role."
  else
    errecho "Failed to list buckets. This should not happen."
    export AWS_ACCESS_KEY_ID=""
    export AWS_SECRET_ACCESS_KEY=""
    export AWS_SESSION_TOKEN=""
    clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn"
 "$policy_arn" "$assume_role_policy_arn"
    return 1
  fi

  local result=0
  export AWS_ACCESS_KEY_ID=""
  export AWS_SECRET_ACCESS_KEY=""

  echo
  echo_repeat "*" 88
  echo "The created resources will now be deleted."
  echo_repeat "*" 88
  echo

  clean_up "$user_name" "$key_name" "$iam_role_name" "$policy_arn" "$policy_arn"
 "$assume_role_policy_arn"

  # shellcheck disable=SC2181
  if [[ ${?} -ne 0 ]]; then
    result=1
  fi

  return $result
}
```

The IAM functions used in this scenario.

```
##############################################################################
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
 (IAM) user already exists.
#
# Parameters:
#       $1 - The name of the IAM user to check.
#
# Returns:
#       0 - If the user already exists.
#       1 - If the user doesn't exist.
##############################################################################
function iam_user_exists() {
  local user_name
  user_name=$1

  # Check whether the IAM user already exists.
  # We suppress all output - we're interested only in the return code.

  local errors
  errors=$(aws iam get-user \
    --user-name "$user_name" 2>&1 >/dev/null)

  local error_code=${?}

  if [[ $error_code -eq 0 ]]; then
    return 0 # 0 in Bash script means true.
  else
    if [[ $errors != *"error"*"(NoSuchEntity)"* ]]; then
      aws_cli_error_log $error_code
      errecho "Error calling iam get-user $errors"
    fi

    return 1 # 1 in Bash script means false.
  fi
}

##############################################################################
# function iam_create_user
#
# This function creates the specified IAM user, unless
```

```
# it already exists.
#
# Parameters:
#       -u user_name  -- The name of the user to create.
#
# Returns:
#       The ARN of the user.
#     And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_user() {
  local user_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user"
    echo "Creates an AWS Identity and Access Management (IAM) user. You must
 supply a username:"
    echo "  -u user_name    The name of the user. It must be unique within the
 account."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "u:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    User name:   $user_name"
  iecho ""

  # If the user already exists, we don't want to try to create it.
  if (iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name already exists in the account."
    return 1
  fi

  response=$(aws iam create-user --user-name "$user_name" \
    --output text \
    --query 'User.Arn')

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-user operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_create_user_access_key
#
# This function creates an IAM access key for the specified user.
#
# Parameters:
#       -u user_name -- The name of the IAM user.
#       [-f file_name] -- The optional file name for the access key output.
#
# Returns:
#       [access_key_id access_key_secret]
#     And:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function iam_create_user_access_key() {
  local user_name file_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user_access_key"
    echo "Creates an AWS Identity and Access Management (IAM) key pair."
    echo "  -u user_name   The name of the IAM user."
    echo "  [-f file_name]   Optional file name for the access key output."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "u:f:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      f) file_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  response=$(aws iam create-access-key \
    --user-name "$user_name" \
    --output text)

  local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-access-key operation failed.$response"
    return 1
  fi

  if [[ -n "$file_name" ]]; then
    echo "$response" >"$file_name"
  fi

  local key_id key_secret
  # shellcheck disable=SC2086
  key_id=$(echo $response | cut -f 2 -d ' ')
  # shellcheck disable=SC2086
  key_secret=$(echo $response | cut -f 4 -d ' ')

  echo "$key_id $key_secret"

  return 0
}

###############################################################################
# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_json -- The assume role policy document.
#
# Returns:
#       The ARN of the role.
#     And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_role() {
  local role_name policy_document response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user_access_key"
```

```
  echo "Creates an AWS Identity and Access Management (IAM) role."
  echo "  -n role_name   The name of the IAM role."
  echo "  -p policy_json -- The assume role policy document."
  echo ""
}

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
  case "${option}" in
    n) role_name="${OPTARG}" ;;
    p) policy_document="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$role_name" ]]; then
  errecho "ERROR: You must provide a role name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$policy_document" ]]; then
  errecho "ERROR: You must provide a policy document with the -p parameter."
  usage
  return 1
fi

response=$(aws iam create-role \
  --role-name "$role_name" \
  --assume-role-policy-document "$policy_document" \
  --output text \
  --query Role.Arn)

local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_create_policy
#
# This function creates an IAM policy.
#
# Parameters:
#       -n policy_name -- The name of the IAM policy.
#       -p policy_json -- The policy document.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_policy() {
  local policy_name policy_document response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_policy"
    echo "Creates an AWS Identity and Access Management (IAM) policy."
    echo "  -n policy_name   The name of the IAM policy."
    echo "  -p policy_json -- The policy document."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:p:h" option; do
    case "${option}" in
      n) policy_name="${OPTARG}" ;;
      p) policy_document="${OPTARG}" ;;
      h)
        usage
```

```
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$policy_name" ]]; then
    errecho "ERROR: You must provide a policy name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$policy_document" ]]; then
    errecho "ERROR: You must provide a policy document with the -p parameter."
    usage
    return 1
  fi

  response=$(aws iam create-policy \
    --policy-name "$policy_name" \
    --policy-document "$policy_document" \
    --output text \
    --query Policy.Arn)

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-policy operation failed.\n$response"
    return 1
  fi

  echo "$response"
}

###############################################################################
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a tole.
```

```
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_attach_role_policy() {
  local role_name policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_attach_role_policy"
    echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM
 role."
    echo "  -n role_name   The name of the IAM role."
    echo "  -p policy_ARN -- The IAM policy document ARN."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:p:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      p) policy_arn="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy ARN with the -p parameter."
    usage
    return 1
  fi

  response=$(aws iam attach-role-policy \
    --role-name "$role_name" \
    --policy-arn "$policy_arn")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a tole.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_detach_role_policy() {
  local role_name policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
```

```
 function usage() {
   echo "function iam_detach_role_policy"
   echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM
role."
   echo "  -n role_name   The name of the IAM role."
   echo "  -p policy_ARN -- The IAM policy document ARN."
   echo ""
 }

 # Retrieve the calling parameters.
 while getopts "n:p:h" option; do
   case "${option}" in
     n) role_name="${OPTARG}" ;;
     p) policy_arn="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$role_name" ]]; then
   errecho "ERROR: You must provide a role name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$policy_arn" ]]; then
   errecho "ERROR: You must provide a policy ARN with the -p parameter."
   usage
   return 1
 fi

 response=$(aws iam detach-role-policy \
   --role-name "$role_name" \
   --policy-arn "$policy_arn")

 local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function iam_delete_policy
#
# This function deletes an IAM policy.
#
# Parameters:
#       -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_policy() {
  local policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_policy"
    echo "Deletes an AWS Identity and Access Management (IAM) policy"
    echo "  -n policy_arn -- The name of the IAM policy arn."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) policy_arn="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
```

```
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy arn with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    Policy arn:  $policy_arn"
  iecho ""

  response=$(aws iam delete-policy \
    --policy-arn "$policy_arn")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
  fi

  iecho "delete-policy response:$response"
  iecho

  return 0
}

###############################################################################
# function iam_delete_role
#
# This function deletes an IAM role.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#
```

```
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_role() {
  local role_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_role"
    echo "Deletes an AWS Identity and Access Management (IAM) role"
    echo "  -n role_name -- The name of the IAM role."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  echo "role_name:$role_name"
  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    Role name:  $role_name"
  iecho ""
```

```
  response=$(aws iam delete-role \
    --role-name "$role_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
  fi

  iecho "delete-role response:$response"
  iecho

  return 0
}

###############################################################################
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#       -u user_name  -- The name of the user.
#       -k access_key -- The access key to delete.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_access_key() {
  local user_name access_key response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_access_key"
    echo "Deletes an AWS Identity and Access Management (IAM) access key for the
 specified IAM user"
    echo "  -u user_name    The name of the user."
    echo "  -k access_key   The access key to delete."
    echo ""
  }
```

```
  # Retrieve the calling parameters.
  while getopts "u:k:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      k) access_key="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    Username:   $user_name"
  iecho "    Access key:   $access_key"
  iecho ""

  response=$(aws iam delete-access-key \
    --user-name "$user_name" \
    --access-key-id "$access_key")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
```

```
    return 1
  fi

  iecho "delete-access-key response:$response"
  iecho

  return 0
}

###############################################################################
# function iam_delete_user
#
# This function deletes the specified IAM user.
#
# Parameters:
#       -u user_name  -- The name of the user to create.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_user() {
  local user_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_user"
    echo "Deletes an AWS Identity and Access Management (IAM) user. You must
 supply a username:"
    echo "  -u user_name    The name of the user."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "u:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
```

```
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    User name:   $user_name"
  iecho ""

  # If the user does not exist, we don't want to try to delete it.
  if (! iam_user_exists "$user_name"); then
    errecho "ERROR: A user with that name does not exist in the account."
    return 1
  fi

  response=$(aws iam delete-user \
    --user-name "$user_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-user operation failed.$response"
    return 1
  fi

  iecho "delete-user response:$response"
  iecho

  return 0
}
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy

- CreateAccessKey
- CreatePolicy
- CreateRole
- CreateUser
- DeleteAccessKey
- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace AwsDoc {
    namespace IAM {

        //! Cleanup by deleting created entities.
        /*!
          \sa DeleteCreatedEntities
          \param client: IAM client.
          \param role: IAM role.
          \param user: IAM user.
          \param policy: IAM policy.
        */
        static bool DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
                                          const Aws::IAM::Model::Role &role,
                                          const Aws::IAM::Model::User &user,
                                          const Aws::IAM::Model::Policy &policy);
```

```
    }

    static const int LIST_BUCKETS_WAIT_SEC = 20;

    static const char ALLOCATION_TAG[] = "example_code";
}

//! Scenario to create an IAM user, create an IAM role, and apply the role to the
 user.
// "IAM access" permissions are needed to run this code.
// "STS assume role" permissions are needed to run this code. (Note: It might be
 necessary to
//    create a custom policy).
/*!
  \sa iamCreateUserAssumeRoleScenario
  \param clientConfig: Aws client configuration.
  \return bool: Successful completion.
*/
bool AwsDoc::IAM::iamCreateUserAssumeRoleScenario(
        const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::IAM::IAMClient client(clientConfig);
    Aws::IAM::Model::User user;
    Aws::IAM::Model::Role role;
    Aws::IAM::Model::Policy policy;

    // 1. Create a user.
    {
        Aws::IAM::Model::CreateUserRequest request;
        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String userName = "iam-demo-user-" +
                               Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetUserName(userName);

        Aws::IAM::Model::CreateUserOutcome outcome = client.CreateUser(request);
        if (!outcome.IsSuccess()) {
            std::cout << "Error creating IAM user " << userName << ":" <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            std::cout << "Successfully created IAM user " << userName <<
 std::endl;
        }
```

```
        user = outcome.GetResult().GetUser();
    }

    // 2. Create a role.
    {
        // Get the IAM user for the current client in order to access its ARN.
        Aws::String iamUserArn;
        {
            Aws::IAM::Model::GetUserRequest request;
            Aws::IAM::Model::GetUserOutcome outcome = client.GetUser(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Error getting Iam user. " <<
                          outcome.GetError().GetMessage() << std::endl;

                DeleteCreatedEntities(client, role, user, policy);
                return false;
            }
            else {
                std::cout << "Successfully retrieved Iam user "
                          << outcome.GetResult().GetUser().GetUserName()
                          << std::endl;
            }

            iamUserArn = outcome.GetResult().GetUser().GetArn();
        }

        Aws::IAM::Model::CreateRoleRequest request;

        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String roleName = "iam-demo-role-" +
                               Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetRoleName(roleName);

        // Build policy document for role.
        Aws::Utils::Document jsonStatement;
        jsonStatement.WithString("Effect", "Allow");

        Aws::Utils::Document jsonPrincipal;
        jsonPrincipal.WithString("AWS", iamUserArn);
        jsonStatement.WithObject("Principal", jsonPrincipal);
        jsonStatement.WithString("Action", "sts:AssumeRole");
        jsonStatement.WithObject("Condition", Aws::Utils::Document());
```

```
        Aws::Utils::Document policyDocument;
        policyDocument.WithString("Version", "2012-10-17");

        Aws::Utils::Array<Aws::Utils::Document> statements(1);
        statements[0] = jsonStatement;
        policyDocument.WithArray("Statement", statements);

        std::cout << "Setting policy for role\n   "
                  << policyDocument.View().WriteCompact() << std::endl;

        // Set role policy document as JSON string.

request.SetAssumeRolePolicyDocument(policyDocument.View().WriteCompact());

        Aws::IAM::Model::CreateRoleOutcome outcome = client.CreateRole(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error creating role. " <<
                      outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully created a role with name " << roleName
                      << std::endl;
        }

        role = outcome.GetResult().GetRole();
    }

    // 3. Create an IAM policy.
    {
        Aws::IAM::Model::CreatePolicyRequest request;
        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String policyName = "iam-demo-policy-" +
                                 Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetPolicyName(policyName);

        // Build IAM policy document.
        Aws::Utils::Document jsonStatement;
        jsonStatement.WithString("Effect", "Allow");
        jsonStatement.WithString("Action", "s3:ListAllMyBuckets");
        jsonStatement.WithString("Resource", "arn:aws:s3:::*");
```

```
        Aws::Utils::Document policyDocument;
        policyDocument.WithString("Version", "2012-10-17");

        Aws::Utils::Array<Aws::Utils::Document> statements(1);
        statements[0] = jsonStatement;
        policyDocument.WithArray("Statement", statements);

        std::cout << "Creating a policy.\n   " <<
policyDocument.View().WriteCompact()
                  << std::endl;

        // Set IAM policy document as JSON string.
        request.SetPolicyDocument(policyDocument.View().WriteCompact());

        Aws::IAM::Model::CreatePolicyOutcome outcome =
client.CreatePolicy(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error creating policy. " <<
                      outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully created a policy with name, " <<
policyName <<
                      "." << std::endl;
        }

        policy = outcome.GetResult().GetPolicy();
    }

    // 4. Assume the new role using the AWS Security Token Service (STS).
    Aws::STS::Model::Credentials credentials;
    {
        Aws::STS::STSClient stsClient(clientConfig);

        Aws::STS::Model::AssumeRoleRequest request;
        request.SetRoleArn(role.GetArn());
        Aws::String uuid = Aws::Utils::UUID::RandomUUID();
        Aws::String roleSessionName = "iam-demo-role-session-" +

Aws::Utils::StringUtils::ToLower(uuid.c_str());
        request.SetRoleSessionName(roleSessionName);
```

```
        Aws::STS::Model::AssumeRoleOutcome assumeRoleOutcome;

        // Repeatedly call AssumeRole, because there is often a delay
        // before the role is available to be assumed.
        // Repeat at most 20 times when access is denied.
        int count = 0;
        while (true) {
            assumeRoleOutcome = stsClient.AssumeRole(request);
            if (!assumeRoleOutcome.IsSuccess()) {
                if (count > 20 ||
                    assumeRoleOutcome.GetError().GetErrorType() !=
                    Aws::STS::STSErrors::ACCESS_DENIED) {
                    std::cerr << "Error assuming role after 20 tries. " <<
                              assumeRoleOutcome.GetError().GetMessage() <<
std::endl;

                    DeleteCreatedEntities(client, role, user, policy);
                    return false;
                }
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                std::cout << "Successfully assumed the role after " << count
                          << " seconds." << std::endl;
                break;
            }
            count++;
        }

        credentials = assumeRoleOutcome.GetResult().GetCredentials();
    }


    // 5. List objects in the bucket (This should fail).
    {
        Aws::S3::S3Client s3Client(
                Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
                                          credentials.GetSecretAccessKey(),
                                          credentials.GetSessionToken()),
                Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
                clientConfig);
        Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
s3Client.ListBuckets();
```

```
        if (!listBucketsOutcome.IsSuccess()) {
            if (listBucketsOutcome.GetError().GetErrorType() !=
                Aws::S3::S3Errors::ACCESS_DENIED) {
                std::cerr << "Could not lists buckets. " <<
                          listBucketsOutcome.GetError().GetMessage() <<
std::endl;
            }
            else {
                std::cout
                        << "Access to list buckets denied because privileges have
not been applied."
                        << std::endl;
            }
        }
        else {
            std::cerr
                    << "Successfully retrieved bucket lists when this should not
happen."
                    << std::endl;
        }
    }

    // 6. Attach the policy to the role.
    {
        Aws::IAM::Model::AttachRolePolicyRequest request;
        request.SetRoleName(role.GetRoleName());
        request.WithPolicyArn(policy.GetArn());

        Aws::IAM::Model::AttachRolePolicyOutcome outcome =
client.AttachRolePolicy(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error creating policy. " <<
                      outcome.GetError().GetMessage() << std::endl;

            DeleteCreatedEntities(client, role, user, policy);
            return false;
        }
        else {
            std::cout << "Successfully attached the policy with name, "
                      << policy.GetPolicyName() <<
                      ", to the role, " << role.GetRoleName() << "." <<
std::endl;
        }
```

```
    }

    int count = 0;
    // 7. List objects in the bucket (this should succeed).
    // Repeatedly call ListBuckets, because there is often a delay
    // before the policy with ListBucket permissions has been applied to the
 role.
    // Repeat at most LIST_BUCKETS_WAIT_SEC times when access is denied.
    while (true) {
        Aws::S3::S3Client s3Client(
                Aws::Auth::AWSCredentials(credentials.GetAccessKeyId(),
                                          credentials.GetSecretAccessKey(),
                                          credentials.GetSessionToken()),
                Aws::MakeShared<Aws::S3::S3EndpointProvider>(ALLOCATION_TAG),
                clientConfig);
        Aws::S3::Model::ListBucketsOutcome listBucketsOutcome =
 s3Client.ListBuckets();
        if (!listBucketsOutcome.IsSuccess()) {
            if ((count > LIST_BUCKETS_WAIT_SEC) ||
                listBucketsOutcome.GetError().GetErrorType() !=
                Aws::S3::S3Errors::ACCESS_DENIED) {
                std::cerr << "Could not lists buckets after " <<
 LIST_BUCKETS_WAIT_SEC << " seconds. " <<
                          listBucketsOutcome.GetError().GetMessage() <<
 std::endl;
                DeleteCreatedEntities(client, role, user, policy);
                return false;
            }

            std::this_thread::sleep_for(std::chrono::seconds(1));
        }
        else {

            std::cout << "Successfully retrieved bucket lists after " << count
                      << " seconds." << std::endl;
            break;
        }
        count++;
    }

    // 8. Delete all the created resources.
    return DeleteCreatedEntities(client, role, user, policy);
}
```

```
bool AwsDoc::IAM::DeleteCreatedEntities(const Aws::IAM::IAMClient &client,
                                        const Aws::IAM::Model::Role &role,
                                        const Aws::IAM::Model::User &user,
                                        const Aws::IAM::Model::Policy &policy) {
    bool result = true;
    if (policy.ArnHasBeenSet()) {
        // Detach the policy from the role.
        {
            Aws::IAM::Model::DetachRolePolicyRequest request;
            request.SetPolicyArn(policy.GetArn());
            request.SetRoleName(role.GetRoleName());

            Aws::IAM::Model::DetachRolePolicyOutcome outcome =
 client.DetachRolePolicy(
                    request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Error Detaching policy from roles. " <<
                          outcome.GetError().GetMessage() << std::endl;
                result = false;
            }
            else {
                std::cout << "Successfully detached the policy with arn "
                          << policy.GetArn()
                          << " from role " << role.GetRoleName() << "." <<
 std::endl;
            }
        }

        // Delete the policy.
        {
            Aws::IAM::Model::DeletePolicyRequest request;
            request.WithPolicyArn(policy.GetArn());

            Aws::IAM::Model::DeletePolicyOutcome outcome =
 client.DeletePolicy(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Error deleting policy. " <<
                          outcome.GetError().GetMessage() << std::endl;
                result = false;
            }
            else {
                std::cout << "Successfully deleted the policy with arn "
                          << policy.GetArn() << std::endl;
            }
```

```
        }

    }

    if (role.RoleIdHasBeenSet()) {
        // Delete the role.
        Aws::IAM::Model::DeleteRoleRequest request;
        request.SetRoleName(role.GetRoleName());

        Aws::IAM::Model::DeleteRoleOutcome outcome = client.DeleteRole(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error deleting role. " <<
                      outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Successfully deleted the role with name "
                      << role.GetRoleName() << std::endl;
        }
    }

    if (user.ArnHasBeenSet()) {
        // Delete the user.
        Aws::IAM::Model::DeleteUserRequest request;
        request.WithUserName(user.GetUserName());

        Aws::IAM::Model::DeleteUserOutcome outcome = client.DeleteUser(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Error deleting user. " <<
                      outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Successfully deleted the user with name "
                      << user.GetUserName() << std::endl;
        }
    }

    return result;
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.

- AttachRolePolicy
- CreateAccessKey
- CreatePolicy
- CreateRole
- CreateUser
- DeleteAccessKey
- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "math/rand"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
```

```
 "github.com/aws/aws-sdk-go-v2/credentials"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/sts"
 "github.com/aws/smithy-go"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/iam/actions"
)

// AssumeRoleScenario shows you how to use the AWS Identity and Access Management
 (IAM)
// service to perform the following actions:
//
//  1. Create a user who has no permissions.
//  2. Create a role that grants permission to list Amazon Simple Storage Service
//     (Amazon S3) buckets for the account.
//  3. Add a policy to let the user assume the role.
//  4. Try and fail to list buckets without permissions.
//  5. Assume the role and list S3 buckets using temporary credentials.
//  6. Delete the policy, role, and user.
type AssumeRoleScenario struct {
 sdkConfig      aws.Config
 accountWrapper actions.AccountWrapper
 policyWrapper  actions.PolicyWrapper
 roleWrapper    actions.RoleWrapper
 userWrapper    actions.UserWrapper
 questioner     demotools.IQuestioner
 helper         IScenarioHelper
 isTestRun      bool
}

// NewAssumeRoleScenario constructs an AssumeRoleScenario instance from a
 configuration.
// It uses the specified config to get an IAM client and create wrappers for the
 actions
// used in the scenario.
func NewAssumeRoleScenario(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) AssumeRoleScenario {
 iamClient := iam.NewFromConfig(sdkConfig)
 return AssumeRoleScenario{
  sdkConfig:      sdkConfig,
  accountWrapper: actions.AccountWrapper{IamClient: iamClient},
```

```
  policyWrapper:  actions.PolicyWrapper{IamClient: iamClient},
  roleWrapper:    actions.RoleWrapper{IamClient: iamClient},
  userWrapper:    actions.UserWrapper{IamClient: iamClient},
  questioner:     questioner,
  helper:         helper,
 }
}

// addTestOptions appends the API options specified in the original configuration
 to
// another configuration. This is used to attach the middleware stubber to
 clients
// that are constructed during the scenario, which is needed for unit testing.
func (scenario AssumeRoleScenario) addTestOptions(scenarioConfig *aws.Config) {
 if scenario.isTestRun {
  scenarioConfig.APIOptions = append(scenarioConfig.APIOptions,
 scenario.sdkConfig.APIOptions...)
 }
}

// Run runs the interactive scenario.
func (scenario AssumeRoleScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo.\n")
   log.Println(r)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the AWS Identity and Access Management (IAM) assume role
 demo.")
 log.Println(strings.Repeat("-", 88))

 user := scenario.CreateUser(ctx)
 accessKey := scenario.CreateAccessKey(ctx, user)
 role := scenario.CreateRoleAndPolicies(ctx, user)
 noPermsConfig := scenario.ListBucketsWithoutPermissions(ctx, accessKey)
 scenario.ListBucketsWithAssumedRole(ctx, noPermsConfig, role)
 scenario.Cleanup(ctx, user, role)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
```

```
}

// CreateUser creates a new IAM user. This user has no permissions.
func (scenario AssumeRoleScenario) CreateUser(ctx context.Context) *types.User {
 log.Println("Let's create an example user with no permissions.")
 userName := scenario.questioner.Ask("Enter a name for the example user:",
 demotools.NotEmpty{})
 user, err := scenario.userWrapper.GetUser(ctx, userName)
 if err != nil {
  panic(err)
 }
 if user == nil {
  user, err = scenario.userWrapper.CreateUser(ctx, userName)
  if err != nil {
   panic(err)
  }
  log.Printf("Created user %v.\n", *user.UserName)
 } else {
  log.Printf("User %v already exists.\n", *user.UserName)
 }
 log.Println(strings.Repeat("-", 88))
 return user
}

// CreateAccessKey creates an access key for the user.
func (scenario AssumeRoleScenario) CreateAccessKey(ctx context.Context, user
 *types.User) *types.AccessKey {
 accessKey, err := scenario.userWrapper.CreateAccessKeyPair(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("Created access key %v for your user.", *accessKey.AccessKeyId)
 log.Println("Waiting a few seconds for your user to be ready...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return accessKey
}

// CreateRoleAndPolicies creates a policy that grants permission to list S3
 buckets for
// the current account and attaches the policy to a newly created role. It also
 adds an
// inline policy to the specified user that grants the user permission to assume
 the role.
```

```
func (scenario AssumeRoleScenario) CreateRoleAndPolicies(ctx context.Context,
 user *types.User) *types.Role {
 log.Println("Let's create a role and policy that grant permission to list S3
 buckets.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 listBucketsRole, err := scenario.roleWrapper.CreateRole(ctx,
 scenario.helper.GetName(), *user.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created role %v.\n", *listBucketsRole.RoleName)
 listBucketsPolicy, err := scenario.policyWrapper.CreatePolicy(
  ctx, scenario.helper.GetName(), []string{"s3:ListAllMyBuckets"},
 "arn:aws:s3:::*")
 if err != nil {
  panic(err)
 }
 log.Printf("Created policy %v.\n", *listBucketsPolicy.PolicyName)
 err = scenario.roleWrapper.AttachRolePolicy(ctx, *listBucketsPolicy.Arn,
 *listBucketsRole.RoleName)
 if err != nil {
  panic(err)
 }
 log.Printf("Attached policy %v to role %v.\n", *listBucketsPolicy.PolicyName,
  *listBucketsRole.RoleName)
 err = scenario.userWrapper.CreateUserPolicy(ctx, *user.UserName,
 scenario.helper.GetName(),
  []string{"sts:AssumeRole"}, *listBucketsRole.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created an inline policy for user %v that lets the user assume the
 role.\n",
  *user.UserName)
 log.Println("Let's give AWS a few seconds to propagate these new resources and
 connections...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return listBucketsRole
}

// ListBucketsWithoutPermissions creates an Amazon S3 client from the user's
 access key
```

```
// credentials and tries to list buckets for the account. Because the user does
 not have
// permission to perform this action, the action fails.
func (scenario AssumeRoleScenario) ListBucketsWithoutPermissions(ctx
 context.Context, accessKey *types.AccessKey) *aws.Config {
 log.Println("Let's try to list buckets without permissions. This should return
 an AccessDenied error.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 noPermsConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *accessKey.AccessKeyId, *accessKey.SecretAccessKey, ""),
  ))
 if err != nil {
  panic(err)
 }

 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&noPermsConfig)

 s3Client := s3.NewFromConfig(noPermsConfig)
 _, err = s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  // The SDK for Go does not model the AccessDenied error, so check ErrorCode
 directly.
  var ae smithy.APIError
  if errors.As(err, &ae) {
   switch ae.ErrorCode() {
   case "AccessDenied":
    log.Println("Got AccessDenied error, which is the expected result because\n"
 +
     "the ListBuckets call was made without permissions.")
   default:
    log.Println("Expected AccessDenied, got something else.")
    panic(err)
   }
  }
 } else {
  log.Println("Expected AccessDenied error when calling ListBuckets without
 permissions,\n" +
   "but the call succeeded. Continuing the example anyway...")
 }
 log.Println(strings.Repeat("-", 88))
 return &noPermsConfig
```

```
}

// ListBucketsWithAssumedRole performs the following actions:
//
//  1. Creates an AWS Security Token Service (AWS STS) client from the config
 created from
//     the user's access key credentials.
//  2. Gets temporary credentials by assuming the role that grants permission to
 list the
//     buckets.
//  3. Creates an Amazon S3 client from the temporary credentials.
//  4. Lists buckets for the account. Because the temporary credentials are
 generated by
//     assuming the role that grants permission, the action succeeds.
func (scenario AssumeRoleScenario) ListBucketsWithAssumedRole(ctx
 context.Context, noPermsConfig *aws.Config, role *types.Role) {
 log.Println("Let's assume the role that grants permission to list buckets and
 try again.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 stsClient := sts.NewFromConfig(*noPermsConfig)
 tempCredentials, err := stsClient.AssumeRole(ctx, &sts.AssumeRoleInput{
  RoleArn:         role.Arn,
  RoleSessionName: aws.String("AssumeRoleExampleSession"),
  DurationSeconds: aws.Int32(900),
 })
 if err != nil {
  log.Printf("Couldn't assume role %v.\n", *role.RoleName)
  panic(err)
 }
 log.Printf("Assumed role %v, got temporary credentials.\n", *role.RoleName)
 assumeRoleConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *tempCredentials.Credentials.AccessKeyId,
   *tempCredentials.Credentials.SecretAccessKey,
   *tempCredentials.Credentials.SessionToken),
  ),
 )
 if err != nil {
  panic(err)
 }

 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&assumeRoleConfig)
```

```
 s3Client := s3.NewFromConfig(assumeRoleConfig)
 result, err := s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  log.Println("Couldn't list buckets with assumed role credentials.")
  panic(err)
 }
 log.Println("Successfully called ListBuckets with assumed role credentials, \n"
 +
  "here are some of them:")
 for i := 0; i < len(result.Buckets) && i < 5; i++ {
  log.Printf("\t%v\n", *result.Buckets[i].Name)
 }
 log.Println(strings.Repeat("-", 88))
}

// Cleanup deletes all resources created for the scenario.
func (scenario AssumeRoleScenario) Cleanup(ctx context.Context, user *types.User,
 role *types.Role) {
 if scenario.questioner.AskBool(
  "Do you want to delete the resources created for this example? (y/n)", "y",
 ) {
  policies, err := scenario.roleWrapper.ListAttachedRolePolicies(ctx,
 *role.RoleName)
  if err != nil {
   panic(err)
  }
  for _, policy := range policies {
   err = scenario.roleWrapper.DetachRolePolicy(ctx, *role.RoleName,
 *policy.PolicyArn)
   if err != nil {
    panic(err)
   }
   err = scenario.policyWrapper.DeletePolicy(ctx, *policy.PolicyArn)
   if err != nil {
    panic(err)
   }
   log.Printf("Detached policy %v from role %v and deleted the policy.\n",
    *policy.PolicyName, *role.RoleName)
  }
  err = scenario.roleWrapper.DeleteRole(ctx, *role.RoleName)
  if err != nil {
   panic(err)
  }
```

```
  log.Printf("Deleted role %v.\n", *role.RoleName)

  userPols, err := scenario.userWrapper.ListUserPolicies(ctx, *user.UserName)
  if err != nil {
   panic(err)
  }
  for _, userPol := range userPols {
   err = scenario.userWrapper.DeleteUserPolicy(ctx, *user.UserName, userPol)
   if err != nil {
    panic(err)
   }
   log.Printf("Deleted policy %v from user %v.\n", userPol, *user.UserName)
  }
  keys, err := scenario.userWrapper.ListAccessKeys(ctx, *user.UserName)
  if err != nil {
   panic(err)
  }
  for _, key := range keys {
   err = scenario.userWrapper.DeleteAccessKey(ctx, *user.UserName,
 *key.AccessKeyId)
   if err != nil {
    panic(err)
   }
   log.Printf("Deleted access key %v from user %v.\n", *key.AccessKeyId,
 *user.UserName)
  }
  err = scenario.userWrapper.DeleteUser(ctx, *user.UserName)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted user %v.\n", *user.UserName)
  log.Println(strings.Repeat("-", 88))
 }

}

// IScenarioHelper abstracts input and wait functions from a scenario so that
 they
// can be mocked for unit testing.
type IScenarioHelper interface {
 GetName() string
 Pause(secs int)
}
```

```
const rMax = 100000

type ScenarioHelper struct {
 Prefix string
 Random *rand.Rand
}

// GetName returns a unique name formed of a prefix and a random number.
func (helper *ScenarioHelper) GetName() string {
 return fmt.Sprintf("%v%v", helper.Prefix, helper.Random.Intn(rMax))
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}
```

Define a struct that wraps account actions.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// GetAccountPasswordPolicy gets the account password policy for the current
 account.
// If no policy has been set, a NoSuchEntityException is error is returned.
```

```
func (wrapper AccountWrapper) GetAccountPasswordPolicy(ctx context.Context)
 (*types.PasswordPolicy, error) {
 var pwPolicy *types.PasswordPolicy
 result, err := wrapper.IamClient.GetAccountPasswordPolicy(ctx,
  &iam.GetAccountPasswordPolicyInput{})
 if err != nil {
  log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
 } else {
  pwPolicy = result.PasswordPolicy
 }
 return pwPolicy, err
}



// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders(ctx context.Context)
 ([]types.SAMLProviderListEntry, error) {
 var providers []types.SAMLProviderListEntry
 result, err := wrapper.IamClient.ListSAMLProviders(ctx,
 &iam.ListSAMLProvidersInput{})
 if err != nil {
  log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
 } else {
  providers = result.SAMLProviderList
 }
 return providers, err
}
```

Define a struct that wraps policy actions.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)
```

```
// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(ctx context.Context, maxPolicies int32)
 ([]types.Policy, error) {
 var policies []types.Policy
 result, err := wrapper.IamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPolicies),
 })
 if err != nil {
  log.Printf("Couldn't list policies. Here's why: %v\n", err)
 } else {
  policies = result.Policies
 }
 return policies, err
}



// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    []string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
}
```

```
// CreatePolicy creates a policy that grants a list of actions to the specified
 resource.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(ctx context.Context, policyName string,
 actions []string,
 resourceArn string) (*types.Policy, error) {
 var policy *types.Policy
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(resourceArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
 resourceArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreatePolicy(ctx, &iam.CreatePolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
 })
 if err != nil {
  log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}



// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(ctx context.Context, policyArn string)
 (*types.Policy, error) {
 var policy *types.Policy
 result, err := wrapper.IamClient.GetPolicy(ctx, &iam.GetPolicyInput{
  PolicyArn: aws.String(policyArn),
 })
```

```
 if err != nil {
  log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}



// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(ctx context.Context, policyArn string)
 error {
 _, err := wrapper.IamClient.DeletePolicy(ctx, &iam.DeletePolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
 }
 return err
}
```

Define a struct that wraps role actions.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}
```

```
// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(ctx context.Context, maxRoles int32)
 ([]types.Role, error) {
 var roles []types.Role
 result, err := wrapper.IamClient.ListRoles(ctx,
  &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
 )
 if err != nil {
  log.Printf("Couldn't list roles. Here's why: %v\n", err)
 } else {
  roles = result.Roles
 }
 return roles, err
}



// CreateRole creates a role that trusts a specified user. The trusted user can
 assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(ctx context.Context, roleName string,
 trustedUserArn string) (*types.Role, error) {
 var role *types.Role
 trustPolicy := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Principal: map[string]string{"AWS": trustedUserArn},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyBytes, err := json.Marshal(trustPolicy)
 if err != nil {
  log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
 trustedUserArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreateRole(ctx, &iam.CreateRoleInput{
```

```
  AssumeRolePolicyDocument: aws.String(string(policyBytes)),
  RoleName:                 aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(ctx context.Context, roleName string)
 (*types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.GetRole(ctx,
  &iam.GetRoleInput{RoleName: aws.String(roleName)})
 if err != nil {
  log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// CreateServiceLinkedRole creates a service-linked role that is owned by the
 specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(ctx context.Context,
 serviceName string, description string) (
 *types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.CreateServiceLinkedRole(ctx,
 &iam.CreateServiceLinkedRoleInput{
  AWSServiceName: aws.String(serviceName),
  Description:    aws.String(description),
 })
 if err != nil {
  log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
 serviceName, err)
 } else {
```

```
  role = result.Role
 }
 return role, err
}



// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(ctx context.Context, roleName
 string) error {
 _, err := wrapper.IamClient.DeleteServiceLinkedRole(ctx,
 &iam.DeleteServiceLinkedRoleInput{
  RoleName: aws.String(roleName)},
 )
 if err != nil {
  log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n",
 roleName, err)
 }
 return err
}



// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(ctx context.Context, policyArn
 string, roleName string) error {
 _, err := wrapper.IamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
 roleName, err)
 }
 return err
}



// ListAttachedRolePolicies lists the policies that are attached to the specified
 role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(ctx context.Context, roleName
 string) ([]types.AttachedPolicy, error) {
 var policies []types.AttachedPolicy
```

```
 result, err := wrapper.IamClient.ListAttachedRolePolicies(ctx,
 &iam.ListAttachedRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
 roleName, err)
 } else {
  policies = result.AttachedPolicies
 }
 return policies, err
}



// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(ctx context.Context, roleName string,
 policyArn string) error {
 _, err := wrapper.IamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName,
 err)
 }
 return err
}



// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(ctx context.Context, roleName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListRolePolicies(ctx,
 &iam.ListRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName,
 err)
 } else {
  policies = result.PolicyNames
```

```
 }
 return policies, err
}



// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(ctx context.Context, roleName string) error
 {
 _, err := wrapper.IamClient.DeleteRole(ctx, &iam.DeleteRoleInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

Define a struct that wraps user actions.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}
```

```
// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(ctx context.Context, maxUsers int32)
 ([]types.User, error) {
 var users []types.User
 result, err := wrapper.IamClient.ListUsers(ctx, &iam.ListUsersInput{
  MaxItems: aws.Int32(maxUsers),
 })
 if err != nil {
  log.Printf("Couldn't list users. Here's why: %v\n", err)
 } else {
  users = result.Users
 }
 return users, err
}



// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.GetUser(ctx, &iam.GetUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NoSuchEntityException:
    log.Printf("User %v does not exist.\n", userName)
    err = nil
   default:
    log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
   }
  }
 } else {
  user = result.User
 }
 return user, err
}
```

```
// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.CreateUser(ctx, &iam.CreateUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
 } else {
  user = result.User
 }
 return user, err
}



// CreateUserPolicy adds an inline policy to a user. This example creates a
 policy that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(ctx context.Context, userName string,
 policyName string, actions []string,
 roleArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(roleArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
 err)
  return err
 }
 _, err = wrapper.IamClient.PutUserPolicy(ctx, &iam.PutUserPolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
```

```
  UserName:       aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName,
 err)
 }
 return err
}



// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(ctx context.Context, userName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListUserPolicies(ctx,
 &iam.ListUserPoliciesInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName,
 err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}



// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(ctx context.Context, userName string,
 policyName string) error {
 _, err := wrapper.IamClient.DeleteUserPolicy(ctx, &iam.DeleteUserPolicyInput{
  PolicyName: aws.String(policyName),
  UserName:   aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName,
 err)
 }
 return err
}
```

```
// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(ctx context.Context, userName string) error
 {
 _, err := wrapper.IamClient.DeleteUser(ctx, &iam.DeleteUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
 }
 return err
}



// CreateAccessKeyPair creates an access key for a user. The returned access key
 contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(ctx context.Context, userName
 string) (*types.AccessKey, error) {
 var key *types.AccessKey
 result, err := wrapper.IamClient.CreateAccessKey(ctx, &iam.CreateAccessKeyInput{
  UserName: aws.String(userName)})
 if err != nil {
  log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
 userName, err)
 } else {
  key = result.AccessKey
 }
 return key, err
}



// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(ctx context.Context, userName string,
 keyId string) error {
 _, err := wrapper.IamClient.DeleteAccessKey(ctx, &iam.DeleteAccessKeyInput{
  AccessKeyId: aws.String(keyId),
  UserName:    aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
```

```
 }
 return err
}



// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(ctx context.Context, userName string)
 ([]types.AccessKeyMetadata, error) {
 var keys []types.AccessKeyMetadata
 result, err := wrapper.IamClient.ListAccessKeys(ctx, &iam.ListAccessKeysInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
 err)
 } else {
  keys = result.AccessKeyMetadata
 }
 return keys, err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM user actions.

```
/*
  To run this Java V2 code example, set up your development environment,
 including your credentials.

  For information, see this documentation topic:

  https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html

  This example performs these operations:

  1. Creates a user that has no permissions.
  2. Creates a role and policy that grants Amazon S3 permissions.
  3. Creates a role.
  4. Grants the user permissions.
  5. Gets temporary credentials by assuming the role.  Creates an Amazon S3
 Service client object with the temporary credentials.
  6. Deletes the resources.
 */

public class IAMScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");
    public static final String PolicyDocument = "{" +
            "  \"Version\": \"2012-10-17\"," +
            "  \"Statement\": [" +
            "    {" +
            "        \"Effect\": \"Allow\"," +
            "        \"Action\": [" +
            "            \"s3:*\"" +
            "       ]," +
```

```
            "       \"Resource\": \"*\"" +
            "    }" +
            "   ]" +
            "}";

   public static String userArn;

   public static void main(String[] args) throws Exception {

       final String usage = """

               Usage:
                   <username> <policyName> <roleName> <roleSessionName>
<bucketName>\s

               Where:
                   username - The name of the IAM user to create.\s
                   policyName - The name of the policy to create.\s
                   roleName - The name of the role to create.\s
                   roleSessionName - The name of the session required for the
assumeRole operation.\s
                   bucketName - The name of the Amazon S3 bucket from which
objects are read.\s
               """;

       if (args.length != 5) {
           System.out.println(usage);
           System.exit(1);
       }

       String userName = args[0];
       String policyName = args[1];
       String roleName = args[2];
       String roleSessionName = args[3];
       String bucketName = args[4];

       Region region = Region.AWS_GLOBAL;
       IamClient iam = IamClient.builder()
               .region(region)
               .build();

       System.out.println(DASHES);
       System.out.println("Welcome to the AWS IAM example scenario.");
       System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println(" 1. Create the IAM user.");
        User createUser = createIAMUser(iam, userName);

        System.out.println(DASHES);
        userArn = createUser.arn();

        AccessKey myKey = createIAMAccessKey(iam, userName);
        String accessKey = myKey.accessKeyId();
        String secretKey = myKey.secretAccessKey();
        String assumeRolePolicyDocument = "{" +
                "\"Version\": \"2012-10-17\"," +
                "\"Statement\": [{" +
                "\"Effect\": \"Allow\"," +
                "\"Principal\": {" +
                " \"AWS\": \"" + userArn + "\"" +
                "}," +
                "\"Action\": \"sts:AssumeRole\"" +
                "}]" +
                "}";

        System.out.println(assumeRolePolicyDocument);
        System.out.println(userName + " was successfully created.");
        System.out.println(DASHES);
        System.out.println("2. Creates a policy.");
        String polArn = createIAMPolicy(iam, policyName);
        System.out.println("The policy " + polArn + " was successfully
created.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Creates a role.");
        TimeUnit.SECONDS.sleep(30);
        String roleArn = createIAMRole(iam, roleName, assumeRolePolicyDocument);
        System.out.println(roleArn + " was successfully created.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Grants the user permissions.");
        attachIAMRolePolicy(iam, roleName, polArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("*** Wait for 30 secs so the resource is available");
        TimeUnit.SECONDS.sleep(30);
        System.out.println("5. Gets temporary credentials by assuming the
role.");
        System.out.println("Perform an Amazon S3 Service operation using the
temporary credentials.");
        assumeRole(roleArn, roleSessionName, bucketName, accessKey, secretKey);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6 Getting ready to delete the AWS resources");
        deleteKey(iam, userName, accessKey);
        deleteRole(iam, roleName, polArn);
        deleteIAMUser(iam, userName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("This IAM Scenario has successfully completed");
        System.out.println(DASHES);
    }

    public static AccessKey createIAMAccessKey(IamClient iam, String user) {
        try {
            CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
                    .userName(user)
                    .build();

            CreateAccessKeyResponse response = iam.createAccessKey(request);
            return response.accessKey();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static User createIAMUser(IamClient iam, String username) {
        try {
            // Create an IamWaiter object
            IamWaiter iamWaiter = iam.waiter();
            CreateUserRequest request = CreateUserRequest.builder()
                    .userName(username)
                    .build();
```

```
            // Wait until the user is created.
            CreateUserResponse response = iam.createUser(request);
            GetUserRequest userRequest = GetUserRequest.builder()
                    .userName(response.user().userName())
                    .build();

            WaiterResponse<GetUserResponse> waitUntilUserExists =
iamWaiter.waitUntilUserExists(userRequest);

waitUntilUserExists.matched().response().ifPresent(System.out::println);
            return response.user();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static String createIAMRole(IamClient iam, String rolename, String
json) {

        try {
            CreateRoleRequest request = CreateRoleRequest.builder()
                    .roleName(rolename)
                    .assumeRolePolicyDocument(json)
                    .description("Created using the AWS SDK for Java")
                    .build();

            CreateRoleResponse response = iam.createRole(request);
            System.out.println("The ARN of the role is " +
response.role().arn());
            return response.role().arn();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createIAMPolicy(IamClient iam, String policyName) {
        try {
```

```
            // Create an IamWaiter object.
            IamWaiter iamWaiter = iam.waiter();
            CreatePolicyRequest request = CreatePolicyRequest.builder()
                    .policyName(policyName)
                    .policyDocument(PolicyDocument).build();

            CreatePolicyResponse response = iam.createPolicy(request);
            GetPolicyRequest polRequest = GetPolicyRequest.builder()
                    .policyArn(response.policy().arn())
                    .build();

            WaiterResponse<GetPolicyResponse> waitUntilPolicyExists =
iamWaiter.waitUntilPolicyExists(polRequest);

waitUntilPolicyExists.matched().response().ifPresent(System.out::println);
            return response.policy().arn();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void attachIAMRolePolicy(IamClient iam, String roleName, String
policyArn) {
        try {
            ListAttachedRolePoliciesRequest request =
ListAttachedRolePoliciesRequest.builder()
                    .roleName(roleName)
                    .build();

            ListAttachedRolePoliciesResponse response =
iam.listAttachedRolePolicies(request);
            List<AttachedPolicy> attachedPolicies = response.attachedPolicies();
            String polArn;
            for (AttachedPolicy policy : attachedPolicies) {
                polArn = policy.policyArn();
                if (polArn.compareTo(policyArn) == 0) {
                    System.out.println(roleName + " policy is already attached to
this role.");
                    return;
                }
            }
```

```
            AttachRolePolicyRequest attachRequest =
AttachRolePolicyRequest.builder()
                    .roleName(roleName)
                    .policyArn(policyArn)
                    .build();

            iam.attachRolePolicy(attachRequest);
            System.out.println("Successfully attached policy " + policyArn + " to
role " + roleName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    // Invoke an Amazon S3 operation using the Assumed Role.
    public static void assumeRole(String roleArn, String roleSessionName, String
bucketName, String keyVal,
            String keySecret) {

        // Use the creds of the new IAM user that was created in this code
example.
        AwsBasicCredentials credentials = AwsBasicCredentials.create(keyVal,
keySecret);
        StsClient stsClient = StsClient.builder()
                .region(Region.US_EAST_1)

.credentialsProvider(StaticCredentialsProvider.create(credentials))
                .build();

        try {
            AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
                    .roleArn(roleArn)
                    .roleSessionName(roleSessionName)
                    .build();

            AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
            Credentials myCreds = roleResponse.credentials();
            String key = myCreds.accessKeyId();
            String secKey = myCreds.secretAccessKey();
            String secToken = myCreds.sessionToken();
```

```
            // List all objects in an Amazon S3 bucket using the temp creds
retrieved by
            // invoking assumeRole.
            Region region = Region.US_EAST_1;
            S3Client s3 = S3Client.builder()
                    .credentialsProvider(

StaticCredentialsProvider.create(AwsSessionCredentials.create(key, secKey,
secToken)))
                    .region(region)
                    .build();

            System.out.println("Created a S3Client using temp credentials.");
            System.out.println("Listing objects in " + bucketName);
            ListObjectsRequest listObjects = ListObjectsRequest.builder()
                    .bucket(bucketName)
                    .build();

            ListObjectsResponse res = s3.listObjects(listObjects);
            List<S3Object> objects = res.contents();
            for (S3Object myValue : objects) {
                System.out.println("The name of the key is " + myValue.key());
                System.out.println("The owner is " + myValue.owner());
            }

        } catch (StsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteRole(IamClient iam, String roleName, String polArn)
{

        try {
            // First the policy needs to be detached.
            DetachRolePolicyRequest rolePolicyRequest =
DetachRolePolicyRequest.builder()
                    .policyArn(polArn)
                    .roleName(roleName)
                    .build();

            iam.detachRolePolicy(rolePolicyRequest);
```

```
            // Delete the policy.
            DeletePolicyRequest request = DeletePolicyRequest.builder()
                    .policyArn(polArn)
                    .build();

            iam.deletePolicy(request);
            System.out.println("*** Successfully deleted " + polArn);

            // Delete the role.
            DeleteRoleRequest roleRequest = DeleteRoleRequest.builder()
                    .roleName(roleName)
                    .build();

            iam.deleteRole(roleRequest);
            System.out.println("*** Successfully deleted " + roleName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteKey(IamClient iam, String username, String
accessKey) {
        try {
            DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()
                    .accessKeyId(accessKey)
                    .userName(username)
                    .build();

            iam.deleteAccessKey(request);
            System.out.println("Successfully deleted access key " + accessKey +
                    " from user " + username);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteIAMUser(IamClient iam, String userName) {
        try {
            DeleteUserRequest request = DeleteUserRequest.builder()
                    .userName(userName)
```

```
                    .build();

            iam.deleteUser(request);
            System.out.println("*** Successfully deleted " + userName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
import {
  CreateUserCommand,
  GetUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "iam_basic_test_username";
const policyName = "iam_basic_test_policy";
const roleName = "iam_basic_test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
```

```
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    }
    throw caught;
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service
 (AWS STS).
  // It's not best practice to use access keys. For more information, see
 https://aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
```

```
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
      return await listBuckets(s3Client);
    } catch (err) {
      if (err instanceof Error && err.name === "InvalidAccessKeyId") {
        throw err;
      }
    }
  });

  // Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
  // is thrown while the user and access keys are still stabilizing.
  const { Role } = await retry(
    {
      intervalInMs: 2000,
      maxRetries: 60,
    },
    () =>
      iamClient.send(
        new CreateRoleCommand({
          AssumeRolePolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
              {
                Effect: "Allow",
```

```
              Principal: {
                // Allow the previously created user to assume this role.
                AWS: User.Arn,
              },
              Action: "sts:AssumeRole",
            },
          ],
        }),
        RoleName: roleName,
      }),
    ),
);

if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: ["s3:ListAllMyBuckets"],
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  }),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
```

```
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 120 }, () =>
  listBuckets(s3Client),
);
```

```
  // Clean up.
  await iamClient.send(
    new DetachRolePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeletePolicyCommand({
      PolicyArn: listBucketPolicy.Arn,
    }),
  );

  await iamClient.send(
    new DeleteRoleCommand({
      RoleName: Role.RoleName,
    }),
  );

  await iamClient.send(
    new DeleteAccessKeyCommand({
      UserName: userName,
      AccessKeyId,
    }),
  );

  await iamClient.send(
    new DeleteUserCommand({
      UserName: userName,
    }),
  );
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }
```

```
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM user actions.

```
suspend fun main(args: Array<String>) {
    val usage = """
    Usage:
        <username> <policyName> <roleName> <roleSessionName> <fileLocation>
 <bucketName>
```

```
    Where:
        username - The name of the IAM user to create.
        policyName - The name of the policy to create.
        roleName - The name of the role to create.
        roleSessionName - The name of the session required for the assumeRole
 operation.
        fileLocation - The file location to the JSON required to create the role
 (see Readme).
        bucketName - The name of the Amazon S3 bucket from which objects are
 read.
    """

    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }

    val userName = args[0]
    val policyName = args[1]
    val roleName = args[2]
    val roleSessionName = args[3]
    val fileLocation = args[4]
    val bucketName = args[5]

    createUser(userName)
    println("$userName was successfully created.")

    val polArn = createPolicy(policyName)
    println("The policy $polArn was successfully created.")

    val roleArn = createRole(roleName, fileLocation)
    println("$roleArn was successfully created.")
    attachRolePolicy(roleName, polArn)

    println("*** Wait for 1 MIN so the resource is available.")
    delay(60000)
    assumeGivenRole(roleArn, roleSessionName, bucketName)

    println("*** Getting ready to delete the AWS resources.")
    deleteRole(roleName, polArn)
    deleteUser(userName)
    println("This IAM Scenario has successfully completed.")
}
```

```
suspend fun createUser(usernameVal: String?): String? {
    val request =
        CreateUserRequest {
            userName = usernameVal
        }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createUser(request)
        return response.user?.userName
    }
}

suspend fun createPolicy(policyNameVal: String?): String {
    val policyDocumentValue = """
    {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "s3:*"
                ],
                "Resource": "*"
            }
        ]
    }
    """.trimIndent()

    val request =
        CreatePolicyRequest {
            policyName = policyNameVal
            policyDocument = policyDocumentValue
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createPolicy(request)
        return response.policy?.arn.toString()
    }
}

suspend fun createRole(
    rolenameVal: String?,
    fileLocation: String?,
```

```
): String? {
    val jsonObject = fileLocation?.let { readJsonSimpleDemo(it) } as JSONObject

    val request =
        CreateRoleRequest {
            roleName = rolenameVal
            assumeRolePolicyDocument = jsonObject.toJSONString()
            description = "Created using the AWS SDK for Kotlin"
        }

    IamClient { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createRole(request)
        return response.role?.arn
    }
}

suspend fun attachRolePolicy(
    roleNameVal: String,
    policyArnVal: String,
) {
    val request =
        ListAttachedRolePoliciesRequest {
            roleName = roleNameVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAttachedRolePolicies(request)
        val attachedPolicies = response.attachedPolicies

        // Ensure that the policy is not attached to this role.
        val checkStatus: Int
        if (attachedPolicies != null) {
            checkStatus = checkMyList(attachedPolicies, policyArnVal)
            if (checkStatus == -1) {
                return
            }
        }

        val policyRequest =
            AttachRolePolicyRequest {
                roleName = roleNameVal
                policyArn = policyArnVal
            }
        iamClient.attachRolePolicy(policyRequest)
```

```
        println("Successfully attached policy $policyArnVal to role
 $roleNameVal")
    }
}

fun checkMyList(
    attachedPolicies: List<AttachedPolicy>,
    policyArnVal: String,
): Int {
    for (policy in attachedPolicies) {
        val polArn = policy.policyArn.toString()

        if (polArn.compareTo(policyArnVal) == 0) {
            println("The policy is already attached to this role.")
            return -1
        }
    }
    return 0
}

suspend fun assumeGivenRole(
    roleArnVal: String?,
    roleSessionNameVal: String?,
    bucketName: String,
) {
    val stsClient = StsClient.fromEnvironment { region = "us-east-1" }
    val roleRequest =
        AssumeRoleRequest {
            roleArn = roleArnVal
            roleSessionName = roleSessionNameVal
        }

    val roleResponse = stsClient.assumeRole(roleRequest)
    val myCreds = roleResponse.credentials
    val key = myCreds?.accessKeyId
    val secKey = myCreds?.secretAccessKey
    val secToken = myCreds?.sessionToken

    val staticCredentials = StaticCredentialsProvider {
        accessKeyId = key
        secretAccessKey = secKey
        sessionToken = secToken
    }
```

```
    // List all objects in an Amazon S3 bucket using the temp creds.
    val s3 = S3Client.fromEnvironment {
        region = "us-east-1"
        credentialsProvider = staticCredentials
    }

    println("Created a S3Client using temp credentials.")
    println("Listing objects in $bucketName")

    val listObjects =
        ListObjectsRequest {
            bucket = bucketName
        }

    val response = s3.listObjects(listObjects)
    response.contents?.forEach { myObject ->
        println("The name of the key is ${myObject.key}")
        println("The owner is ${myObject.owner}")
    }
}

suspend fun deleteRole(
    roleNameVal: String,
    polArn: String,
) {
    val iam = IamClient.fromEnvironment { region = "AWS_GLOBAL" }

    // First the policy needs to be detached.
    val rolePolicyRequest =
        DetachRolePolicyRequest {
            policyArn = polArn
            roleName = roleNameVal
        }

    iam.detachRolePolicy(rolePolicyRequest)

    // Delete the policy.
    val request =
        DeletePolicyRequest {
            policyArn = polArn
        }

    iam.deletePolicy(request)
    println("*** Successfully deleted $polArn")
```

```
    // Delete the role.
    val roleRequest =
        DeleteRoleRequest {
            roleName = roleNameVal
        }

    iam.deleteRole(roleRequest)
    println("*** Successfully deleted $roleNameVal")
}

suspend fun deleteUser(userNameVal: String) {
    val iam = IamClient.fromEnvironment { region = "AWS_GLOBAL" }
    val request =
        DeleteUserRequest {
            userName = userNameVal
        }

    iam.deleteUser(request)
    println("*** Successfully deleted $userNameVal")
}

@Throws(java.lang.Exception::class)
fun readJsonSimpleDemo(filename: String): Any? {
    val reader = FileReader(filename)
    val jsonParser = JSONParser()
    return jsonParser.parse(reader)
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser

- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace Iam\Basics;

require 'vendor/autoload.php';

use Aws\Credentials\Credentials;
use Aws\S3\Exception\S3Exception;
use Aws\S3\S3Client;
use Aws\Sts\StsClient;
use Iam\IAMService;

echo("\n");
echo("---------------------------------------\n");
print("Welcome to the IAM getting started demo using PHP!\n");
echo("---------------------------------------\n");

$uuid = uniqid();
$service = new IAMService();

$user = $service->createUser("iam_demo_user_$uuid");
echo "Created user with the arn: {$user['Arn']}\n";

$key = $service->createAccessKey($user['UserName']);
$assumeRolePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
```

```
                    \"Action\": \"sts:AssumeRole\"
                }]
            }";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
 $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";

$listAllBucketsPolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
 $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

$service->attachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);

$inlinePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"sts:AssumeRole\",
                    \"Resource\": \"{$assumeRoleRole['Arn']}\"}]
}";
$inlinePolicy = $service->createUserPolicy("iam_demo_inline_policy_$uuid",
 $inlinePolicyDocument, $user['UserName']);
//First, fail to list the buckets with the user
$credentials = new Credentials($key['AccessKeyId'], $key['SecretAccessKey']);
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $credentials]);
try {
    $s3Client->listBuckets([
    ]);
    echo "this should not run";
} catch (S3Exception $exception) {
    echo "successfully failed!\n";
}

$stsClient = new StsClient(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $credentials]);
```

```
sleep(10);
$assumedRole = $stsClient->assumeRole([
    'RoleArn' => $assumeRoleRole['Arn'],
    'RoleSessionName' => "DemoAssumeRoleSession_$uuid",
]);
$assumedCredentials = [
    'key' => $assumedRole['Credentials']['AccessKeyId'],
    'secret' => $assumedRole['Credentials']['SecretAccessKey'],
    'token' => $assumedRole['Credentials']['SessionToken'],
];
$s3Client = new S3Client(['region' => 'us-west-2', 'version' => 'latest',
 'credentials' => $assumedCredentials]);
try {
    $s3Client->listBuckets([]);
    echo "this should now run!\n";
} catch (S3Exception $exception) {
    echo "this should now not fail\n";
}

$service->detachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);
$deletePolicy = $service->deletePolicy($listAllBucketsPolicy['Arn']);
echo "Delete policy: {$listAllBucketsPolicy['PolicyName']}\n";
$deletedRole = $service->deleteRole($assumeRoleRole['Arn']);
echo "Deleted role: {$assumeRoleRole['RoleName']}\n";
$deletedKey = $service->deleteAccessKey($key['AccessKeyId'], $user['UserName']);
$deletedUser = $service->deleteUser($user['UserName']);
echo "Delete user: {$user['UserName']}\n";
```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole

- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
import json
import sys
import time
from uuid import uuid4

import boto3
from botocore.exceptions import ClientError


def progress_bar(seconds):
    """Shows a simple progress bar in the command window."""
    for _ in range(seconds):
        time.sleep(1)
        print(".", end="")
        sys.stdout.flush()
    print()


def setup(iam_resource):
    """
    Creates a new user with no permissions.
```

```
    Creates an access key pair for the user.
    Creates a role with a policy that lets the user assume the role.
    Creates a policy that allows listing Amazon S3 buckets.
    Attaches the policy to the role.
    Creates an inline policy for the user that lets the user assume the role.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM)
resource
                         that has permissions to create users, roles, and
policies
                         in the account.
    :return: The newly created user, user key, and role.
    """
    try:
        user = iam_resource.create_user(UserName=f"demo-user-{uuid4()}")
        print(f"Created user {user.name}.")
    except ClientError as error:
        print(
            f"Couldn't create a user for the demo. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    try:
        user_key = user.create_access_key_pair()
        print(f"Created access key pair for user.")
    except ClientError as error:
        print(
            f"Couldn't create access keys for user {user.name}. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    print(f"Wait for user to be ready.", end="")
    progress_bar(10)

    try:
        role = iam_resource.create_role(
            RoleName=f"demo-role-{uuid4()}",
            AssumeRolePolicyDocument=json.dumps(
                {
                    "Version":"2012-10-17",
                    "Statement": [
                        {
```

```
                            "Effect": "Allow",
                            "Principal": {"AWS": user.arn},
                            "Action": "sts:AssumeRole",
                        }
                    ],
                }
            ),
        )
        print(f"Created role {role.name}.")
    except ClientError as error:
        print(
            f"Couldn't create a role for the demo. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    try:
        policy = iam_resource.create_policy(
            PolicyName=f"demo-policy-{uuid4()}",
            PolicyDocument=json.dumps(
                {
                    "Version":"2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": "s3:ListAllMyBuckets",
                            "Resource": "arn:aws:s3:::*",
                        }
                    ],
                }
            ),
        )
        role.attach_policy(PolicyArn=policy.arn)
        print(f"Created policy {policy.policy_name} and attached it to the
role.")
    except ClientError as error:
        print(
            f"Couldn't create a policy and attach it to role {role.name}. Here's
why: "
            f"{error.response['Error']['Message']}"
        )
        raise

    try:
```

```
        user.create_policy(
            PolicyName=f"demo-user-policy-{uuid4()}",
            PolicyDocument=json.dumps(
                {
                    "Version":"2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": "sts:AssumeRole",
                            "Resource": role.arn,
                        }
                    ],
                }
            ),
        )
        print(
            f"Created an inline policy for {user.name} that lets the user assume
 "
            f"the role."
        )
    except ClientError as error:
        print(
            f"Couldn't create an inline policy for user {user.name}. Here's why:
 "
            f"{error.response['Error']['Message']}"
        )
        raise

    print("Give AWS time to propagate these new resources and connections.",
 end="")
    progress_bar(10)

    return user, user_key, role


def show_access_denied_without_role(user_key):
    """
    Shows that listing buckets without first assuming the role is not allowed.

    :param user_key: The key of the user created during setup. This user does not
                     have permission to list buckets in the account.
    """
    print(f"Try to list buckets without first assuming the role.")
    s3_denied_resource = boto3.resource(
```

```
        "s3", aws_access_key_id=user_key.id,
 aws_secret_access_key=user_key.secret
    )
    try:
        for bucket in s3_denied_resource.buckets.all():
            print(bucket.name)
        raise RuntimeError("Expected to get AccessDenied error when listing
 buckets!")
    except ClientError as error:
        if error.response["Error"]["Code"] == "AccessDenied":
            print("Attempt to list buckets with no permissions: AccessDenied.")
        else:
            raise


def list_buckets_from_assumed_role(user_key, assume_role_arn, session_name):
    """
    Assumes a role that grants permission to list the Amazon S3 buckets in the
 account.
    Uses the temporary credentials from the role to list the buckets that are
 owned
    by the assumed role's account.

    :param user_key: The access key of a user that has permission to assume the
 role.
    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                            grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    """
    sts_client = boto3.client(
        "sts", aws_access_key_id=user_key.id,
 aws_secret_access_key=user_key.secret
    )
    try:
        response = sts_client.assume_role(
            RoleArn=assume_role_arn, RoleSessionName=session_name
        )
        temp_credentials = response["Credentials"]
        print(f"Assumed role {assume_role_arn} and got temporary credentials.")
    except ClientError as error:
        print(
            f"Couldn't assume role {assume_role_arn}. Here's why: "
            f"{error.response['Error']['Message']}"
        )
```

```
        raise

    # Create an S3 resource that can access the account with the temporary
 credentials.
    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )
    print(f"Listing buckets for the assumed role's account:")
    try:
        for bucket in s3_resource.buckets.all():
            print(bucket.name)
    except ClientError as error:
        print(
            f"Couldn't list buckets for the account. Here's why: "
            f"{error.response['Error']['Message']}"
        )
        raise




def teardown(user, role):
    """
    Removes all resources created during setup.

    :param user: The demo user.
    :param role: The demo role.
    """
    try:
        for attached in role.attached_policies.all():
            policy_name = attached.policy_name
            role.detach_policy(PolicyArn=attached.arn)
            attached.delete()
            print(f"Detached and deleted {policy_name}.")
        role.delete()
        print(f"Deleted {role.name}.")
    except ClientError as error:
        print(
            "Couldn't detach policy, delete policy, or delete role. Here's why: "
            f"{error.response['Error']['Message']}"
        )
```

```
            raise

        try:
            for user_pol in user.policies.all():
                user_pol.delete()
                print("Deleted inline user policy.")
            for key in user.access_keys.all():
                key.delete()
                print("Deleted user's access key.")
            user.delete()
            print(f"Deleted {user.name}.")
        except ClientError as error:
            print(
                "Couldn't delete user policy or delete user. Here's why: "
                f"{error.response['Error']['Message']}"
            )


def usage_demo():
    """Drives the demonstration."""
    print("-" * 88)
    print(f"Welcome to the IAM create user and assume role demo.")
    print("-" * 88)
    iam_resource = boto3.resource("iam")
    user = None
    role = None
    try:
        user, user_key, role = setup(iam_resource)
        print(f"Created {user.name} and {role.name}.")
        show_access_denied_without_role(user_key)
        list_buckets_from_assumed_role(user_key, role.arn,
 "AssumeRoleDemoSession")
    except Exception:
        print("Something went wrong!")
    finally:
        if user is not None and role is not None:
            teardown(user, role)
        print("Thanks for watching!")


if __name__ == "__main__":
    usage_demo()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user and a role that grants permission to list Amazon S3 buckets. The user has rights only to assume the role. After assuming the role, use temporary credentials to list buckets for the account.

```
# Wraps the scenario actions.
class ScenarioCreateUserAssumeRole
  attr_reader :iam_client

  # @param [Aws::IAM::Client] iam_client: The AWS IAM client.
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
```

```
  end

  # Waits for the specified number of seconds.
  #
  # @param duration [Integer] The number of seconds to wait.
  def wait(duration)
    puts('Give AWS time to propagate resources...')
    sleep(duration)
  end

  # Creates a user.
  #
  # @param user_name [String] The name to give the user.
  # @return [Aws::IAM::User] The newly created user.
  def create_user(user_name)
    user = @iam_client.create_user(user_name: user_name).user
    @logger.info("Created demo user named #{user.user_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info('Tried and failed to create demo user.')
    @logger.info("\t#{e.code}: #{e.message}")
    @logger.info("\nCan't continue the demo without a user!")
    raise
  else
    user
  end

  # Creates an access key for a user.
  #
  # @param user [Aws::IAM::User] The user that owns the key.
  # @return [Aws::IAM::AccessKeyPair] The newly created access key.
  def create_access_key_pair(user)
    user_key = @iam_client.create_access_key(user_name:
user.user_name).access_key
    @logger.info("Created accesskey pair for user #{user.user_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info("Couldn't create access keys for user #{user.user_name}.")
    @logger.info("\t#{e.code}: #{e.message}")
    raise
  else
    user_key
  end

  # Creates a role that can be assumed by a user.
  #
```

```
 # @param role_name [String] The name to give the role.
 # @param user [Aws::IAM::User] The user who is granted permission to assume the
role.
 # @return [Aws::IAM::Role] The newly created role.
 def create_role(role_name, user)
   trust_policy = {
     Version: '2012-10-17',
     Statement: [{
       Effect: 'Allow',
       Principal: { 'AWS': user.arn },
       Action: 'sts:AssumeRole'
     }]
   }.to_json
   role = @iam_client.create_role(
     role_name: role_name,
     assume_role_policy_document: trust_policy
   ).role
   @logger.info("Created role #{role.role_name}.")
 rescue Aws::Errors::ServiceError => e
   @logger.info("Couldn't create a role for the demo. Here's why: ")
   @logger.info("\t#{e.code}: #{e.message}")
   raise
 else
   role
 end

 # Creates a policy that grants permission to list S3 buckets in the account,
and
 # then attaches the policy to a role.
 #
 # @param policy_name [String] The name to give the policy.
 # @param role [Aws::IAM::Role] The role that the policy is attached to.
 # @return [Aws::IAM::Policy] The newly created policy.
 def create_and_attach_role_policy(policy_name, role)
   policy_document = {
     Version: '2012-10-17',
     Statement: [{
       Effect: 'Allow',
       Action: 's3:ListAllMyBuckets',
       Resource: 'arn:aws:s3:::*'
     }]
   }.to_json
   policy = @iam_client.create_policy(
     policy_name: policy_name,
```

```
      policy_document: policy_document
    ).policy
    @iam_client.attach_role_policy(
      role_name: role.role_name,
      policy_arn: policy.arn
    )
    @logger.info("Created policy #{policy.policy_name} and attached it to role
#{role.role_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info("Couldn't create a policy and attach it to role
#{role.role_name}. Here's why: ")
    @logger.info("\t#{e.code}: #{e.message}")
    raise
  end

  # Creates an inline policy for a user that lets the user assume a role.
  #
  # @param policy_name [String] The name to give the policy.
  # @param user [Aws::IAM::User] The user that owns the policy.
  # @param role [Aws::IAM::Role] The role that can be assumed.
  # @return [Aws::IAM::UserPolicy] The newly created policy.
  def create_user_policy(policy_name, user, role)
    policy_document = {
      Version: '2012-10-17',
      Statement: [{
        Effect: 'Allow',
        Action: 'sts:AssumeRole',
        Resource: role.arn
      }]
    }.to_json
    @iam_client.put_user_policy(
      user_name: user.user_name,
      policy_name: policy_name,
      policy_document: policy_document
    )
    puts("Created an inline policy for #{user.user_name} that lets the user
assume role #{role.role_name}.")
  rescue Aws::Errors::ServiceError => e
    @logger.info("Couldn't create an inline policy for user #{user.user_name}.
Here's why: ")
    @logger.info("\t#{e.code}: #{e.message}")
    raise
  end
```

```
 # Creates an Amazon S3 resource with specified credentials. This is separated
into a
 # factory function so that it can be mocked for unit testing.
 #
 # @param credentials [Aws::Credentials] The credentials used by the Amazon S3
resource.
 def create_s3_resource(credentials)
   Aws::S3::Resource.new(client: Aws::S3::Client.new(credentials: credentials))
 end

 # Lists the S3 buckets for the account, using the specified Amazon S3 resource.
 # Because the resource uses credentials with limited access, it may not be able
to
 # list the S3 buckets.
 #
 # @param s3_resource [Aws::S3::Resource] An Amazon S3 resource.
 def list_buckets(s3_resource)
   count = 10
   s3_resource.buckets.each do |bucket|
     @logger.info "\t#{bucket.name}"
     count -= 1
     break if count.zero?
   end
 rescue Aws::Errors::ServiceError => e
   if e.code == 'AccessDenied'
     puts('Attempt to list buckets with no permissions: AccessDenied.')
   else
     @logger.info("Couldn't list buckets for the account. Here's why: ")
     @logger.info("\t#{e.code}: #{e.message}")
     raise
   end
 end

 # Creates an AWS Security Token Service (AWS STS) client with specified
credentials.
 # This is separated into a factory function so that it can be mocked for unit
testing.
 #
 # @param key_id [String] The ID of the access key used by the STS client.
 # @param key_secret [String] The secret part of the access key used by the STS
client.
 def create_sts_client(key_id, key_secret)
   Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)
 end
```

```
 # Gets temporary credentials that can be used to assume a role.
 #
 # @param role_arn [String] The ARN of the role that is assumed when these
credentials
 #                          are used.
 # @param sts_client [AWS::STS::Client] An AWS STS client.
 # @return [Aws::AssumeRoleCredentials] The credentials that can be used to
assume the role.
 def assume_role(role_arn, sts_client)
   credentials = Aws::AssumeRoleCredentials.new(
     client: sts_client,
     role_arn: role_arn,
     role_session_name: 'create-use-assume-role-scenario'
   )
   @logger.info("Assumed role '#{role_arn}', got temporary credentials.")
   credentials
 end

 # Deletes a role. If the role has policies attached, they are detached and
 # deleted before the role is deleted.
 #
 # @param role_name [String] The name of the role to delete.
 def delete_role(role_name)
   @iam_client.list_attached_role_policies(role_name:
role_name).attached_policies.each do |policy|
     @iam_client.detach_role_policy(role_name: role_name, policy_arn:
policy.policy_arn)
     @iam_client.delete_policy(policy_arn: policy.policy_arn)
     @logger.info("Detached and deleted policy #{policy.policy_name}.")
   end
   @iam_client.delete_role({ role_name: role_name })
   @logger.info("Role deleted: #{role_name}.")
 rescue Aws::Errors::ServiceError => e
   @logger.info("Couldn't detach policies and delete role #{role.name}. Here's
why:")
   @logger.info("\t#{e.code}: #{e.message}")
   raise
 end

 # Deletes a user. If the user has inline policies or access keys, they are
deleted
 # before the user is deleted.
 #
```

```
  # @param user [Aws::IAM::User] The user to delete.
  def delete_user(user_name)
    user = @iam_client.list_access_keys(user_name: user_name).access_key_metadata
    user.each do |key|
      @iam_client.delete_access_key({ access_key_id: key.access_key_id,
 user_name: user_name })
      @logger.info("Deleted access key #{key.access_key_id} for user
 '#{user_name}'.")
    end

    @iam_client.delete_user(user_name: user_name)
    @logger.info("Deleted user '#{user_name}'.")
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting user '#{user_name}': #{e.message}")
  end
end

# Runs the IAM create a user and assume a role scenario.
def run_scenario(scenario)
  puts('-' * 88)
  puts('Welcome to the IAM create a user and assume a role demo!')
  puts('-' * 88)
  user = scenario.create_user("doc-example-user-#{Random.uuid}")
  user_key = scenario.create_access_key_pair(user)
  scenario.wait(10)
  role = scenario.create_role("doc-example-role-#{Random.uuid}", user)
  scenario.create_and_attach_role_policy("doc-example-role-policy-
#{Random.uuid}", role)
  scenario.create_user_policy("doc-example-user-policy-#{Random.uuid}", user,
 role)
  scenario.wait(10)
  puts('Try to list buckets with credentials for a user who has no permissions.')
  puts('Expect AccessDenied from this call.')
  scenario.list_buckets(
    scenario.create_s3_resource(Aws::Credentials.new(user_key.access_key_id,
 user_key.secret_access_key))
  )
  puts('Now, assume the role that grants permission.')
  temp_credentials = scenario.assume_role(
    role.arn, scenario.create_sts_client(user_key.access_key_id,
 user_key.secret_access_key)
  )
  puts('Here are your buckets:')
  scenario.list_buckets(scenario.create_s3_resource(temp_credentials))
```

```
  puts("Deleting role '#{role.role_name}' and attached policies.")
  scenario.delete_role(role.role_name)
  puts("Deleting user '#{user.user_name}', policies, and keys.")
  scenario.delete_user(user.user_name)
  puts('Thanks for watching!')
  puts('-' * 88)
rescue Aws::Errors::ServiceError => e
  puts('Something went wrong with the demo.')
  puts("\t#{e.code}: #{e.message}")
end

run_scenario(ScenarioCreateUserAssumeRole.new(Aws::IAM::Client.new)) if
 $PROGRAM_NAME == __FILE__
```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
use aws_config::meta::region::RegionProviderChain;
use aws_sdk_iam::Error as iamError;
use aws_sdk_iam::{config::Credentials as iamCredentials, config::Region, Client
 as iamClient};
use aws_sdk_s3::Client as s3Client;
use aws_sdk_sts::Client as stsClient;
use tokio::time::{sleep, Duration};
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), iamError> {
    let (client, uuid, list_all_buckets_policy_document, inline_policy_document)
 =
        initialize_variables().await;

    if let Err(e) = run_iam_operations(
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
    .await
    {
        println!("{:?}", e);
    };

    Ok(())
}

async fn initialize_variables() -> (iamClient, String, String, String) {
```

```
    let region_provider = RegionProviderChain::first_try(Region::new("us-
west-2"));

    let shared_config =
 aws_config::from_env().region(region_provider).load().await;
    let client = iamClient::new(&shared_config);
    let uuid = Uuid::new_v4().to_string();

    let list_all_buckets_policy_document = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
    }"
    .to_string();
    let inline_policy_document = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"sts:AssumeRole\",
                    \"Resource\": \"{}\"}]
    }"
    .to_string();

    (
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
}

async fn run_iam_operations(
    client: iamClient,
    uuid: String,
    list_all_buckets_policy_document: String,
    inline_policy_document: String,
) -> Result<(), iamError> {
    let user = iam_service::create_user(&client, &format!("{}{}",
 "iam_demo_user_", uuid)).await?;
    println!("Created the user with the name: {}", user.user_name());
    let key = iam_service::create_access_key(&client, user.user_name()).await?;
```

```
    let assume_role_policy_document = "{
        \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }"
    .to_string()
    .replace("{}", user.arn());

    let assume_role_role = iam_service::create_role(
        &client,
        &format!("{}{}", "iam_demo_role_", uuid),
        &assume_role_policy_document,
    )
    .await?;
    println!("Created the role with the ARN: {}", assume_role_role.arn());

    let list_all_buckets_policy = iam_service::create_policy(
        &client,
        &format!("{}{}", "iam_demo_policy_", uuid),
        &list_all_buckets_policy_document,
    )
    .await?;
    println!(
        "Created policy: {}",
        list_all_buckets_policy.policy_name.as_ref().unwrap()
    );

    let attach_role_policy_result =
        iam_service::attach_role_policy(&client, &assume_role_role,
&list_all_buckets_policy)
            .await?;
    println!(
        "Attached the policy to the role: {:?}",
        attach_role_policy_result
    );

    let inline_policy_name = format!("{}{}", "iam_demo_inline_policy_", uuid);
    let inline_policy_document = inline_policy_document.replace("{}",
assume_role_role.arn());
    iam_service::create_user_policy(&client, &user, &inline_policy_name,
&inline_policy_document)
```

```
        .await?;
    println!("Created inline policy.");

    //First, fail to list the buckets with the user.
    let creds = iamCredentials::from_keys(key.access_key_id(),
key.secret_access_key(), None);
    let fail_config = aws_config::from_env()
        .credentials_provider(creds.clone())
        .load()
        .await;
    println!("Fail config: {:?}", fail_config);
    let fail_client: s3Client = s3Client::new(&fail_config);
    match fail_client.list_buckets().send().await {
        Ok(e) => {
            println!("This should not run. {:?}", e);
        }
        Err(e) => {
            println!("Successfully failed with error: {:?}", e)
        }
    }

    let sts_config = aws_config::from_env()
        .credentials_provider(creds.clone())
        .load()
        .await;
    let sts_client: stsClient = stsClient::new(&sts_config);
    sleep(Duration::from_secs(10)).await;
    let assumed_role = sts_client
        .assume_role()
        .role_arn(assume_role_role.arn())
        .role_session_name(format!("iam_demo_assumerole_session_{uuid}"))
        .send()
        .await;
    println!("Assumed role: {:?}", assumed_role);
    sleep(Duration::from_secs(10)).await;

    let assumed_credentials = iamCredentials::from_keys(
        assumed_role
            .as_ref()
            .unwrap()
            .credentials
            .as_ref()
            .unwrap()
            .access_key_id(),
```

```
        assumed_role
            .as_ref()
            .unwrap()
            .credentials
            .as_ref()
            .unwrap()
            .secret_access_key(),
        Some(
            assumed_role
                .as_ref()
                .unwrap()
                .credentials
                .as_ref()
                .unwrap()
                .session_token
                .clone(),
        ),
    );

    let succeed_config = aws_config::from_env()
        .credentials_provider(assumed_credentials)
        .load()
        .await;
    println!("succeed config: {:?}", succeed_config);
    let succeed_client: s3Client = s3Client::new(&succeed_config);
    sleep(Duration::from_secs(10)).await;
    match succeed_client.list_buckets().send().await {
        Ok(_) => {
            println!("This should now run successfully.")
        }
        Err(e) => {
            println!("This should not run. {:?}", e);
            panic!()
        }
    }

    //Clean up.
    iam_service::detach_role_policy(
        &client,
        assume_role_role.role_name(),
        list_all_buckets_policy.arn().unwrap_or_default(),
    )
    .await?;
    iam_service::delete_policy(&client, list_all_buckets_policy).await?;
```

```
    iam_service::delete_role(&client, &assume_role_role).await?;
    println!("Deleted role {}", assume_role_role.role_name());
    iam_service::delete_access_key(&client, &user, &key).await?;
    println!("Deleted key for {}", key.user_name());
    iam_service::delete_user_policy(&client, &user, &inline_policy_name).await?;
    println!("Deleted inline user policy: {}", inline_policy_name);
    iam_service::delete_user(&client, &user).await?;
    println!("Deleted user {}", user.user_name());

    Ok(())
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy
  - PutUserPolicy

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for IAM using AWS SDKs

The following code examples demonstrate how to perform individual IAM actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the IAM API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for IAM using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the AWS Identity and Access Management API Reference.

Examples

- Use AddClientIdToOpenIdConnectProvider with a CLI
- Use AddRoleToInstanceProfile with a CLI
- Use AddUserToGroup with a CLI
- Use AttachGroupPolicy with a CLI
- Use AttachRolePolicy with an AWS SDK or CLI
- Use AttachUserPolicy with an AWS SDK or CLI
- Use ChangePassword with a CLI
- Use CreateAccessKey with an AWS SDK or CLI
- Use CreateAccountAlias with an AWS SDK or CLI
- Use CreateGroup with an AWS SDK or CLI
- Use CreateInstanceProfile with an AWS SDK or CLI
- Use CreateLoginProfile with a CLI
- Use CreateOpenIdConnectProvider with a CLI
- Use CreatePolicy with an AWS SDK or CLI
- Use CreatePolicyVersion with an AWS SDK or CLI
- Use CreateRole with an AWS SDK or CLI
- Use CreateSAMLProvider with an AWS SDK or CLI
- Use CreateServiceLinkedRole with an AWS SDK or CLI
- Use CreateUser with an AWS SDK or CLI
- Use CreateVirtualMfaDevice with a CLI
- Use DeactivateMfaDevice with a CLI
- Use DeleteAccessKey with an AWS SDK or CLI
- Use DeleteAccountAlias with an AWS SDK or CLI

- Use DeleteAccountPasswordPolicy with a CLI
- Use DeleteGroup with an AWS SDK or CLI
- Use DeleteGroupPolicy with a CLI
- Use DeleteInstanceProfile with an AWS SDK or CLI
- Use DeleteLoginProfile with a CLI
- Use DeleteOpenIdConnectProvider with a CLI
- Use DeletePolicy with an AWS SDK or CLI
- Use DeletePolicyVersion with an AWS SDK or CLI
- Use DeleteRole with an AWS SDK or CLI
- Use DeleteRolePermissionsBoundary with a CLI
- Use DeleteRolePolicy with an AWS SDK or CLI
- Use DeleteSAMLProvider with an AWS SDK or CLI
- Use DeleteServerCertificate with an AWS SDK or CLI
- Use DeleteServiceLinkedRole with an AWS SDK or CLI
- Use DeleteSigningCertificate with a CLI
- Use DeleteUser with an AWS SDK or CLI
- Use DeleteUserPermissionsBoundary with a CLI
- Use DeleteUserPolicy with an AWS SDK or CLI
- Use DeleteVirtualMfaDevice with a CLI
- Use DetachGroupPolicy with a CLI
- Use DetachRolePolicy with an AWS SDK or CLI
- Use DetachUserPolicy with an AWS SDK or CLI
- Use EnableMfaDevice with a CLI
- Use GenerateCredentialReport with an AWS SDK or CLI
- Use GenerateServiceLastAccessedDetails with a CLI
- Use GetAccessKeyLastUsed with an AWS SDK or CLI
- Use GetAccountAuthorizationDetails with an AWS SDK or CLI
- Use GetAccountPasswordPolicy with an AWS SDK or CLI
- Use GetAccountSummary with an AWS SDK or CLI

- Use GetContextKeysForCustomPolicy with a CLI
- Use GetContextKeysForPrincipalPolicy with a CLI
- Use GetCredentialReport with an AWS SDK or CLI
- Use GetGroup with a CLI
- Use GetGroupPolicy with a CLI
- Use GetInstanceProfile with a CLI
- Use GetLoginProfile with a CLI
- Use GetOpenIdConnectProvider with a CLI
- Use GetPolicy with an AWS SDK or CLI
- Use GetPolicyVersion with an AWS SDK or CLI
- Use GetRole with an AWS SDK or CLI
- Use GetRolePolicy with a CLI
- Use GetSamlProvider with a CLI
- Use GetServerCertificate with an AWS SDK or CLI
- Use GetServiceLastAccessedDetails with a CLI
- Use GetServiceLastAccessedDetailsWithEntities with a CLI
- Use GetServiceLinkedRoleDeletionStatus with an AWS SDK or CLI
- Use GetUser with an AWS SDK or CLI
- Use GetUserPolicy with a CLI
- Use ListAccessKeys with an AWS SDK or CLI
- Use ListAccountAliases with an AWS SDK or CLI
- Use ListAttachedGroupPolicies with a CLI
- Use ListAttachedRolePolicies with an AWS SDK or CLI
- Use ListAttachedUserPolicies with a CLI
- Use ListEntitiesForPolicy with a CLI
- Use ListGroupPolicies with a CLI
- Use ListGroups with an AWS SDK or CLI
- Use ListGroupsForUser with a CLI
- Use ListInstanceProfiles with a CLI

- Use ListInstanceProfilesForRole with a CLI
- Use ListMfaDevices with a CLI
- Use ListOpenIdConnectProviders with a CLI
- Use ListPolicies with an AWS SDK or CLI
- Use ListPolicyVersions with an AWS SDK or CLI
- Use ListRolePolicies with an AWS SDK or CLI
- Use ListRoleTags with a CLI
- Use ListRoles with an AWS SDK or CLI
- Use ListSAMLProviders with an AWS SDK or CLI
- Use ListServerCertificates with an AWS SDK or CLI
- Use ListSigningCertificates with a CLI
- Use ListUserPolicies with an AWS SDK or CLI
- Use ListUserTags with a CLI
- Use ListUsers with an AWS SDK or CLI
- Use ListVirtualMfaDevices with a CLI
- Use PutGroupPolicy with a CLI
- Use PutRolePermissionsBoundary with a CLI
- Use PutRolePolicy with an AWS SDK or CLI
- Use PutUserPermissionsBoundary with a CLI
- Use PutUserPolicy with an AWS SDK or CLI
- Use RemoveClientIdFromOpenIdConnectProvider with a CLI
- Use RemoveRoleFromInstanceProfile with a CLI
- Use RemoveUserFromGroup with a CLI
- Use ResyncMfaDevice with a CLI
- Use SetDefaultPolicyVersion with an AWS SDK or CLI
- Use TagRole with a CLI
- Use TagUser with a CLI
- Use UntagRole with a CLI
- Use UntagUser with a CLI

- Use UpdateAccessKey with an AWS SDK or CLI
- Use UpdateAccountPasswordPolicy with a CLI
- Use UpdateAssumeRolePolicy with a CLI
- Use UpdateGroup with a CLI
- Use UpdateLoginProfile with a CLI
- Use UpdateOpenIdConnectProviderThumbprint with a CLI
- Use UpdateRole with a CLI
- Use UpdateRoleDescription with a CLI
- Use UpdateSamlProvider with a CLI
- Use UpdateServerCertificate with an AWS SDK or CLI
- Use UpdateSigningCertificate with a CLI
- Use UpdateUser with an AWS SDK or CLI
- Use UploadServerCertificate with an AWS SDK or CLI
- Use UploadSigningCertificate with a CLI

Use `AddClientIdToOpenIdConnectProvider` with a CLI

The following code examples show how to use `AddClientIdToOpenIdConnectProvider`.

CLI

AWS CLI

To add a client ID (audience) to an Open-ID Connect (OIDC) provider

The following `add-client-id-to-open-id-connect-provider` example adds the client ID `my-application-ID` to the OIDC provider named `server.example.com`.

```
aws iam add-client-id-to-open-id-connect-provider \
    --client-id my-application-ID \
    --open-id-connect-provider-arn arn:aws:iam::123456789012:oidc-provider/
server.example.com
```

This command produces no output.

To create an OIDC provider, use the `create-open-id-connect-provider` command.

For more information, see Creating OpenID Connect (OIDC) identity providers in the *AWS IAM User Guide*.

- For API details, see AddClientIdToOpenIdConnectProvider in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command adds the client ID (or audience) `my-application-ID` to the existing OIDC provider named `server.example.com`.

```
Add-IAMClientIDToOpenIDConnectProvider -ClientID "my-application-ID"
 -OpenIDConnectProviderARN "arn:aws:iam::123456789012:oidc-provider/
server.example.com"
```

- For API details, see AddClientIdToOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command adds the client ID (or audience) `my-application-ID` to the existing OIDC provider named `server.example.com`.

```
Add-IAMClientIDToOpenIDConnectProvider -ClientID "my-application-ID"
 -OpenIDConnectProviderARN "arn:aws:iam::123456789012:oidc-provider/
server.example.com"
```

- For API details, see AddClientIdToOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AddRoleToInstanceProfile` with a CLI

The following code examples show how to use `AddRoleToInstanceProfile`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Creating a container service for virtual machine instances
- Getting started with managed streaming
- Run CPU stress tests on virtual machine instances using fault injection

CLI

AWS CLI

To add a role to an instance profile

The following `add-role-to-instance-profile` example adds the role named `S3Access` to the instance profile named `Webserver`.

```
aws iam add-role-to-instance-profile \
    --role-name S3Access \
    --instance-profile-name Webserver
```

This command produces no output.

To create an instance profile, use the `create-instance-profile` command.

For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *AWS IAM User Guide*.

- For API details, see AddRoleToInstanceProfile in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command adds the role named `S3Access` to an existing instance profile named `webserver`. To create the instance profile, use the `New-IAMInstanceProfile` command. After you create the instance profile and associate it with a role using this command, you can attach it to an EC2 instance. To do that, use the `New-EC2Instance` cmdlet with either the `InstanceProfile_Arn` or the `InstanceProfile-Name` parameter to launch the new instance.

```
Add-IAMRoleToInstanceProfile -RoleName "S3Access" -InstanceProfileName
 "webserver"
```

- For API details, see AddRoleToInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command adds the role named `S3Access` to an existing instance profile named `webserver`. To create the instance profile, use the `New-IAMInstanceProfile` command. After you create the instance profile and associate it with a role using this command, you can attach it to an EC2 instance. To do that, use the `New-EC2Instance` cmdlet with either the `InstanceProfile_Arn` or the `InstanceProfile-Name` parameter to launch the new instance.

```
Add-IAMRoleToInstanceProfile -RoleName "S3Access" -InstanceProfileName
 "webserver"
```

- For API details, see AddRoleToInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AddUserToGroup` with a CLI

The following code examples show how to use `AddUserToGroup`.

CLI

AWS CLI

To add a user to an IAM group

The following `add-user-to-group` example adds an IAM user named Bob to the IAM group named `Admins`.

```
aws iam add-user-to-group \
    --user-name Bob \
```

```
    --group-name Admins
```

This command produces no output.

For more information, see Adding and removing users in an IAM user group in the *AWS IAM User Guide*.

- For API details, see AddUserToGroup in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command adds the user named Bob to the group named Admins.

```
Add-IAMUserToGroup -UserName "Bob" -GroupName "Admins"
```

- For API details, see AddUserToGroup in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command adds the user named Bob to the group named Admins.

```
Add-IAMUserToGroup -UserName "Bob" -GroupName "Admins"
```

- For API details, see AddUserToGroup in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AttachGroupPolicy` with a CLI

The following code examples show how to use `AttachGroupPolicy`.

CLI

AWS CLI

To attach a managed policy to an IAM group

The following `attach-group-policy` example attaches the AWS managed policy named `ReadOnlyAccess` to the IAM group named `Finance`.

```
aws iam attach-group-policy \
    --policy-arn arn:aws:iam::aws:policy/ReadOnlyAccess \
    --group-name Finance
```

This command produces no output.

For more information, see Managed policies and inline policies in the *AWS IAM User Guide*.

- For API details, see AttachGroupPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example attaches the customer managed policy named `TesterPolicy` to the IAM group `Testers`. The users in that group are immediately affected by the permissions defined in the default version of that policy.

```
Register-IAMGroupPolicy -GroupName Testers -PolicyArn
 arn:aws:iam::123456789012:policy/TesterPolicy
```

Example 2: This example attaches the AWS managed policy named `AdministratorAccess` to the IAM group `Admins`. The users in that group are immediately affected by the permissions defined in the latest version of that policy.

```
Register-IAMGroupPolicy -GroupName Admins -PolicyArn arn:aws:iam::aws:policy/
AdministratorAccess
```

- For API details, see AttachGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example attaches the customer managed policy named `TesterPolicy` to the IAM group `Testers`. The users in that group are immediately affected by the permissions defined in the default version of that policy.

```
Register-IAMGroupPolicy -GroupName Testers -PolicyArn
 arn:aws:iam::123456789012:policy/TesterPolicy
```

Example 2: This example attaches the AWS managed policy named `AdministratorAccess` to the IAM group `Admins`. The users in that group are immediately affected by the permissions defined in the latest version of that policy.

```
Register-IAMGroupPolicy -GroupName Admins -PolicyArn arn:aws:iam::aws:policy/
AdministratorAccess
```

- For API details, see AttachGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AttachRolePolicy` with an AWS SDK or CLI

The following code examples show how to use `AttachRolePolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Configure container service connectivity
- Create a container task for the serverless launch type
- Create a rest API with function proxy integration
- Creating a container service for virtual machine instances
- Creating a managed monitoring workspace
- Creating a monitoring dashboard with function name as a variable
- Creating your first serverless function
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with workflow orchestration
- Manage roles

- Moving hardcoded secrets to secure secret storage
- Run CPU stress tests on virtual machine instances using fault injection
- Setting up systems management
- Using property variables in monitoring dashboards to monitor multiple serverless functions
- Work with Streams and Time-to-Live

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
 AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see AttachRolePolicy in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function iam_attach_role_policy
#
# This function attaches an IAM policy to a tole.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function iam_attach_role_policy() {
  local role_name policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_attach_role_policy"
    echo "Attaches an AWS Identity and Access Management (IAM) policy to an IAM
 role."
    echo "  -n role_name   The name of the IAM role."
```

```
  echo "  -p policy_ARN -- The IAM policy document ARN."
  echo ""
}

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
  case "${option}" in
    n) role_name="${OPTARG}" ;;
    p) policy_arn="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$role_name" ]]; then
  errecho "ERROR: You must provide a role name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$policy_arn" ]]; then
  errecho "ERROR: You must provide a policy ARN with the -p parameter."
  usage
  return 1
fi

response=$(aws iam attach-role-policy \
  --role-name "$role_name" \
  --policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports attach-role-policy operation failed.\n$response"
  return 1
```

```
  fi

  echo "$response"

  return 0
}
```

- For API details, see AttachRolePolicy in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::attachRolePolicy(const Aws::String &roleName,
                                   const Aws::String &policyArn,
                                   const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::ListAttachedRolePoliciesRequest list_request;
    list_request.SetRoleName(roleName);

    bool done = false;
    while (!done) {
        auto list_outcome = iam.ListAttachedRolePolicies(list_request);
        if (!list_outcome.IsSuccess()) {
            std::cerr << "Failed to list attached policies of role " <<
                      roleName << ": " << list_outcome.GetError().GetMessage() <<
                      std::endl;
            return false;
        }

        const auto &policies = list_outcome.GetResult().GetAttachedPolicies();
        if (std::any_of(policies.cbegin(), policies.cend(),
                        [=](const Aws::IAM::Model::AttachedPolicy &policy) {
```

```
                                return policy.GetPolicyArn() == policyArn;
                            })) {
                std::cout << "Policy " << policyArn <<
                          " is already attached to role " << roleName << std::endl;
                return true;
            }

            done = !list_outcome.GetResult().GetIsTruncated();
            list_request.SetMarker(list_outcome.GetResult().GetMarker());
        }

        Aws::IAM::Model::AttachRolePolicyRequest request;
        request.SetRoleName(roleName);
        request.SetPolicyArn(policyArn);

        Aws::IAM::Model::AttachRolePolicyOutcome outcome =
     iam.AttachRolePolicy(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to attach policy " << policyArn << " to role " <<
                      roleName << ": " << outcome.GetError().GetMessage() <<
     std::endl;
        }
        else {
            std::cout << "Successfully attached policy " << policyArn << " to role "
     <<
                      roleName << std::endl;
        }

        return outcome.IsSuccess();
    }
```

- For API details, see AttachRolePolicy in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To attach a managed policy to an IAM role

The following `attach-role-policy` example attaches the AWS managed policy named `ReadOnlyAccess` to the IAM role named `ReadOnlyRole`.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/ReadOnlyAccess \
    --role-name ReadOnlyRole
```

This command produces no output.

For more information, see Managed policies and inline policies in the *AWS IAM User Guide*.

- For API details, see AttachRolePolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// AttachRolePolicy attaches a policy to a role.
```

```
func (wrapper RoleWrapper) AttachRolePolicy(ctx context.Context, policyArn
 string, roleName string) error {
 _, err := wrapper.IamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
 roleName, err)
 }
 return err
}
```

- For API details, see AttachRolePolicy in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.AttachRolePolicyRequest;
import software.amazon.awssdk.services.iam.model.AttachedPolicy;
import software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesRequest;
import
 software.amazon.awssdk.services.iam.model.ListAttachedRolePoliciesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AttachRolePolicy {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <roleName> <policyArn>\s

                Where:
                    roleName - A role name that you can obtain from the AWS
 Management Console.\s
                    policyArn - A policy ARN that you can obtain from the AWS
 Management Console.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String roleName = args[0];
        String policyArn = args[1];

        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        attachIAMRolePolicy(iam, roleName, policyArn);
        iam.close();
    }

    public static void attachIAMRolePolicy(IamClient iam, String roleName, String
 policyArn) {
        try {
            ListAttachedRolePoliciesRequest request =
 ListAttachedRolePoliciesRequest.builder()
                    .roleName(roleName)
                    .build();
```

```
            ListAttachedRolePoliciesResponse response =
 iam.listAttachedRolePolicies(request);
            List<AttachedPolicy> attachedPolicies = response.attachedPolicies();

            // Ensure that the policy is not attached to this role
            String polArn = "";
            for (AttachedPolicy policy : attachedPolicies) {
                polArn = policy.policyArn();
                if (polArn.compareTo(policyArn) == 0) {
                    System.out.println(roleName + " policy is already attached to
 this role.");
                    return;
                }
            }

            AttachRolePolicyRequest attachRequest =
 AttachRolePolicyRequest.builder()
                    .roleName(roleName)
                    .policyArn(policyArn)
                    .build();

            iam.attachRolePolicy(attachRequest);

            System.out.println("Successfully attached policy " + policyArn +
                    " to role " + roleName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        System.out.println("Done");
    }
}
```

- For API details, see AttachRolePolicy in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Attach the policy.

```
import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see AttachRolePolicy in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
        process.exit();
      }
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see AttachRolePolicy in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun attachIAMRolePolicy(
    roleNameVal: String,
    policyArnVal: String,
) {
    val request =
        ListAttachedRolePoliciesRequest {
            roleName = roleNameVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAttachedRolePolicies(request)
        val attachedPolicies = response.attachedPolicies

        // Ensure that the policy is not attached to this role.
        val checkStatus: Int
        if (attachedPolicies != null) {
            checkStatus = checkList(attachedPolicies, policyArnVal)
            if (checkStatus == -1) {
                return
            }
        }

        val policyRequest =
            AttachRolePolicyRequest {
                roleName = roleNameVal
                policyArn = policyArnVal
            }
        iamClient.attachRolePolicy(policyRequest)
        println("Successfully attached policy $policyArnVal to role
 $roleNameVal")
    }
}
```

```
fun checkList(
    attachedPolicies: List<AttachedPolicy>,
    policyArnVal: String,
): Int {
    for (policy in attachedPolicies) {
        val polArn = policy.policyArn.toString()

        if (polArn.compareTo(policyArnVal) == 0) {
            println("The policy is already attached to this role.")
            return -1
        }
    }
    return 0
}
```

- For API details, see AttachRolePolicy in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$assumeRolePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
 $assumeRolePolicyDocument);
```

```
echo "Created role: {$assumeRoleRole['RoleName']}\n";

$listAllBucketsPolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
 $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

$service->attachRolePolicy($assumeRoleRole['RoleName'],
 $listAllBucketsPolicy['Arn']);

    public function attachRolePolicy($roleName, $policyArn)
    {
        return $this->customWaiter(function () use ($roleName, $policyArn) {
            $this->iamClient->attachRolePolicy([
                'PolicyArn' => $policyArn,
                'RoleName' => $roleName,
            ]);
        });
    }
```

- For API details, see AttachRolePolicy in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example attaches the AWS managed policy named `SecurityAudit` to the IAM role `CoSecurityAuditors`. The users who assume that role are immediately affected by the permissions defined in the latest version of that policy.

```
Register-IAMRolePolicy -RoleName CoSecurityAuditors -PolicyArn
 arn:aws:iam::aws:policy/SecurityAudit
```

- For API details, see AttachRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example attaches the AWS managed policy named `SecurityAudit` to the IAM role `CoSecurityAuditors`. The users who assume that role are immediately affected by the permissions defined in the latest version of that policy.

```
Register-IAMRolePolicy -RoleName CoSecurityAuditors -PolicyArn
 arn:aws:iam::aws:policy/SecurityAudit
```

- For API details, see AttachRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Attach a policy to a role using the Boto3 Policy object.

```
def attach_to_role(role_name, policy_arn):
    """
    Attaches a policy to a role.

    :param role_name: The name of the role. **Note** this is the name, not the
 ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Policy(policy_arn).attach_role(RoleName=role_name)
        logger.info("Attached policy %s to role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to role %s.", policy_arn,
 role_name)
        raise
```

Attach a policy to a role using the Boto3 Role object.

```
def attach_policy(role_name, policy_arn):
    """
    Attaches a policy to a role.

    :param role_name: The name of the role. **Note** this is the name, not the
 ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Role(role_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to role %s.", policy_arn,
 role_name)
        raise
```

- For API details, see AttachRolePolicy in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, attaches, and detaches role policies.

```
# Manages policies in AWS Identity and Access Management (IAM)
class RolePolicyManager
  # Initialize with an AWS IAM client
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
```

```
    @logger = logger
    @logger.progname = 'PolicyManager'
  end

  # Creates a policy
  #
  # @param policy_name [String] The name of the policy
  # @param policy_document [Hash] The policy document
  # @return [String] The policy ARN if successful, otherwise nil
  def create_policy(policy_name, policy_document)
    response = @iam_client.create_policy(
      policy_name: policy_name,
      policy_document: policy_document.to_json
    )
    response.policy.arn
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating policy: #{e.message}")
    nil
  end

  # Fetches an IAM policy by its ARN
  # @param policy_arn [String] the ARN of the IAM policy to retrieve
  # @return [Aws::IAM::Types::GetPolicyResponse] the policy object if found
  def get_policy(policy_arn)
    response = @iam_client.get_policy(policy_arn: policy_arn)
    policy = response.policy
    @logger.info("Got policy '#{policy.policy_name}'. Its ID is:
#{policy.policy_id}.")
    policy
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Couldn't get policy '#{policy_arn}'. The policy does not
exist.")
    raise
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't get policy '#{policy_arn}'. Here's why: #{e.code}:
#{e.message}")
    raise
  end

  # Attaches a policy to a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
```

```
  def attach_policy_to_role(role_name, policy_arn)
    @iam_client.attach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error attaching policy to role: #{e.message}")
    false
  end

  # Lists policy ARNs attached to a role
  #
  # @param role_name [String] The name of the role
  # @return [Array<String>] List of policy ARNs
  def list_attached_policy_arns(role_name)
    response = @iam_client.list_attached_role_policies(role_name: role_name)
    response.attached_policies.map(&:policy_arn)
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing policies attached to role: #{e.message}")
    []
  end

  # Detaches a policy from a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def detach_policy_from_role(role_name, policy_arn)
    @iam_client.detach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error detaching policy from role: #{e.message}")
    false
  end
end
```

- For API details, see AttachRolePolicy in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn attach_role_policy(
    client: &iamClient,
    role: &Role,
    policy: &Policy,
) -> Result<AttachRolePolicyOutput, SdkError<AttachRolePolicyError>> {
    client
        .attach_role_policy()
        .role_name(role.role_name())
        .policy_arn(policy.arn().unwrap_or_default())
        .send()
        .await
}
```

- For API details, see AttachRolePolicy in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->attachrolepolicy(
          iv_rolename = iv_role_name
          iv_policyarn = iv_policy_arn ).
```

```
        MESSAGE 'Policy attached to role successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Role or policy does not exist.' TYPE 'E'.
      CATCH /aws1/cx_iamlimitexceededex.
        MESSAGE 'Policy attachment limit exceeded.' TYPE 'E'.
    ENDTRY.
```

- For API details, see AttachRolePolicy in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func attachRolePolicy(role: String, policyArn: String) async throws {
        let input = AttachRolePolicyInput(
            policyArn: policyArn,
            roleName: role
        )
        do {
            _ = try await client.attachRolePolicy(input: input)
        } catch {
            print("ERROR: Attaching a role policy:", dump(error))
            throw error
        }
    }
```

- For API details, see AttachRolePolicy in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AttachUserPolicy` with an AWS SDK or CLI

The following code examples show how to use `AttachUserPolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create read-only and read-write users

CLI

AWS CLI

To attach a managed policy to an IAM user

The following `attach-user-policy` example attaches the AWS managed policy named `AdministratorAccess` to the IAM user named `Alice`.

```
aws iam attach-user-policy \
    --policy-arn arn:aws:iam::aws:policy/AdministratorAccess \
    --user-name Alice
```

This command produces no output.

For more information, see Managed policies and inline policies in the *AWS IAM User Guide*.

- For API details, see AttachUserPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example attaches the AWS managed policy named `AmazonCognitoPowerUser` to the IAM user Bob. The user is immediately affected by the permissions defined in the latest version of that policy.

```
Register-IAMUserPolicy -UserName Bob -PolicyArn arn:aws:iam::aws:policy/
AmazonCognitoPowerUser
```

- For API details, see AttachUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example attaches the AWS managed policy named `AmazonCognitoPowerUser` to the IAM user Bob. The user is immediately affected by the permissions defined in the latest version of that policy.

```
Register-IAMUserPolicy -UserName Bob -PolicyArn arn:aws:iam::aws:policy/
AmazonCognitoPowerUser
```

- For API details, see AttachUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def attach_policy(user_name, policy_arn):
    """
    Attaches a policy to a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to user %s.", policy_arn,
 user_name)
        raise
```

- For API details, see AttachUserPolicy in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Attaches a policy to a user
#
# @param user_name [String] The name of the user
# @param policy_arn [String] The Amazon Resource Name (ARN) of the policy
# @return [Boolean] true if successful, false otherwise
def attach_policy_to_user(user_name, policy_arn)
  @iam_client.attach_user_policy(
    user_name: user_name,
    policy_arn: policy_arn
  )
  true
rescue Aws::IAM::Errors::ServiceError => e
  @logger.error("Error attaching policy to user: #{e.message}")
  false
end
```

- For API details, see AttachUserPolicy in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn attach_user_policy(
    client: &iamClient,
    user_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .attach_user_policy()
        .user_name(user_name)
        .policy_arn(policy_arn)
        .send()
        .await?;

    Ok(())
}
```

- For API details, see AttachUserPolicy in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->attachuserpolicy(
```

```
          iv_username = iv_user_name
          iv_policyarn = iv_policy_arn ).
        MESSAGE 'Policy attached to user successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'User or policy does not exist.' TYPE 'E'.
      CATCH /aws1/cx_iamlimitexceededex.
        MESSAGE 'Policy attachment limit exceeded.' TYPE 'E'.
    ENDTRY.
```

- For API details, see AttachUserPolicy in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use ChangePassword with a CLI

The following code examples show how to use `ChangePassword`.

CLI

AWS CLI

To change the password for your IAM user

The following `change-password` example changes the password for your IAM user. To change the password, we recommend using the `--cli-input-json` parameter to pass a JSON file that contains your old and new passwords. Using this method, you can use strong passwords with non-alphanumeric characters. It can be difficult to use passwords with non-alphanumeric characters when you pass them as command line parameters. To use the `--cli-input-json` parameter, start by using the `change-password` command with the `--generate-cli-skeleton` parameter, as in the following example.

```
aws iam change-password \
    --generate-cli-skeleton > change-password.json
```

The previous command creates a JSON file called change-password.json that you can use to fill in your old and new passwords.

Next, to change your password, use the `change-password` command again, this time passing the `--cli-input-json` parameter to specify your JSON file. The following `change-password` command uses the `--cli-input-json` parameter with a JSON file called change-password.json.

```
aws iam change-password \
    --cli-input-json file://change-password.json
```

Contents of `change-password.json`:

```
{
    "OldPassword": "3s0K_;xh4~8XXI",
    "NewPassword": "]35d/{pB9Fo9wJ"
}
```

This command produces no output.

This command can be called by IAM users only. If this command is called using AWS account (root) credentials, the command returns an `InvalidUserType` error.

For more information, see How an IAM user changes their own password in the *AWS IAM User Guide*.

- For API details, see ChangePassword in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command changes the password for the user that is running the command. This command can be called by IAM users only. If this command is called when you are signed-in with AWS account (root) credentials, the command returns an `InvalidUserType` error.

```
Edit-IAMPassword -OldPassword "MyOldP@ssw0rd" -NewPassword "MyNewP@ssw0rd"
```

- For API details, see ChangePassword in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command changes the password for the user that is running the command. This command can be called by IAM users only. If this command is called when you are signed-in with AWS account (root) credentials, the command returns an `InvalidUserType` error.

```
Edit-IAMPassword -OldPassword "MyOldP@ssw0rd" -NewPassword "MyNewP@ssw0rd"
```

- For API details, see ChangePassword in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateAccessKey` with an AWS SDK or CLI

The following code examples show how to use `CreateAccessKey`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Create read-only and read-write users
- Manage access keys

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Create an IAM access key for a user.
/// </summary>
```

```
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
 CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;

    }
```

- For API details, see CreateAccessKey in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function iam_create_user_access_key
#
# This function creates an IAM access key for the specified user.
```

```
#
# Parameters:
#       -u user_name -- The name of the IAM user.
#       [-f file_name] -- The optional file name for the access key output.
#
# Returns:
#       [access_key_id access_key_secret]
#     And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_user_access_key() {
  local user_name file_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user_access_key"
    echo "Creates an AWS Identity and Access Management (IAM) key pair."
    echo "  -u user_name   The name of the IAM user."
    echo "  [-f file_name]   Optional file name for the access key output."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "u:f:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      f) file_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
```

```
    usage
    return 1
  fi

  response=$(aws iam create-access-key \
    --user-name "$user_name" \
    --output text)

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-access-key operation failed.$response"
    return 1
  fi

  if [[ -n "$file_name" ]]; then
    echo "$response" >"$file_name"
  fi

  local key_id key_secret
  # shellcheck disable=SC2086
  key_id=$(echo $response | cut -f 2 -d ' ')
  # shellcheck disable=SC2086
  key_secret=$(echo $response | cut -f 4 -d ' ')

  echo "$key_id $key_secret"

  return 0
}
```

- For API details, see CreateAccessKey in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
Aws::String AwsDoc::IAM::createAccessKey(const Aws::String &userName,
                                         const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::CreateAccessKeyRequest request;
    request.SetUserName(userName);

    Aws::String result;
    Aws::IAM::Model::CreateAccessKeyOutcome outcome =
 iam.CreateAccessKey(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating access key for IAM user " << userName
                  << ":" << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        const auto &accessKey = outcome.GetResult().GetAccessKey();
        std::cout << "Successfully created access key for IAM user " <<
                  userName << std::endl << "  aws_access_key_id = " <<
                  accessKey.GetAccessKeyId() << std::endl <<
                  " aws_secret_access_key = " << accessKey.GetSecretAccessKey()
 <<
                  std::endl;
        result = accessKey.GetAccessKeyId();
    }

    return result;
}
```

- For API details, see CreateAccessKey in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To create an access key for an IAM user

The following `create-access-key` example creates an access key (access key ID and secret access key) for the IAM user named Bob.

```
aws iam create-access-key \
```

```
    --user-name Bob
```

Output:

```
{
    "AccessKey": {
        "UserName": "Bob",
        "Status": "Active",
        "CreateDate": "2015-03-09T18:39:23.411Z",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY",
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE"
    }
}
```

Store the secret access key in a secure location. If it is lost, it cannot be recovered, and you must create a new access key.

For more information, see Managing access keys for IAM users in the *AWS IAM User Guide*.

- For API details, see CreateAccessKey in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
```

```
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// CreateAccessKeyPair creates an access key for a user. The returned access key
 contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(ctx context.Context, userName
 string) (*types.AccessKey, error) {
 var key *types.AccessKey
 result, err := wrapper.IamClient.CreateAccessKey(ctx, &iam.CreateAccessKeyInput{
  UserName: aws.String(userName)})
 if err != nil {
  log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
 userName, err)
 } else {
  key = result.AccessKey
 }
 return key, err
}
```

- For API details, see CreateAccessKey in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.CreateAccessKeyRequest;
import software.amazon.awssdk.services.iam.model.CreateAccessKeyResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateAccessKey {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                   <user>\s

                Where:
                   user - An AWS IAM user that you can obtain from the AWS
 Management Console.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String user = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        String keyId = createIAMAccessKey(iam, user);
        System.out.println("The Key Id is " + keyId);
        iam.close();
    }
```

```
    public static String createIAMAccessKey(IamClient iam, String user) {
        try {
            CreateAccessKeyRequest request = CreateAccessKeyRequest.builder()
                    .userName(user)
                    .build();

            CreateAccessKeyResponse response = iam.createAccessKey(request);
            return response.accessKey().accessKeyId();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see CreateAccessKey in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the access key.

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
```

```
  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateAccessKey in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateAccessKey in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createIAMAccessKey(user: String?): String {
    val request =
        CreateAccessKeyRequest {
            userName = user
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createAccessKey(request)
        return response.accessKey?.accessKeyId.toString()
    }
}
```

- For API details, see CreateAccessKey in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new access key and secret access key pair and assigns it to the user `David`. Ensure that you save the `AccessKeyId` and `SecretAccessKey` values to a file because this is the only time you can obtain the `SecretAccessKey`. You cannot retrieve it later. If you lose the secret key, you must create a new access key pair.

```
New-IAMAccessKey -UserName David
```

Output:

```
AccessKeyId     : AKIAIOSFODNN7EXAMPLE
CreateDate      : 4/13/2015 1:00:42 PM
```

```
SecretAccessKey : wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Status          : Active
UserName        : David
```

- For API details, see CreateAccessKey in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new access key and secret access key pair and assigns it to the user `David`. Ensure that you save the `AccessKeyId` and `SecretAccessKey` values to a file because this is the only time you can obtain the `SecretAccessKey`. You cannot retrieve it later. If you lose the secret key, you must create a new access key pair.

```
New-IAMAccessKey -UserName David
```

Output:

```
AccessKeyId     : AKIAIOSFODNN7EXAMPLE
CreateDate      : 4/13/2015 1:00:42 PM
SecretAccessKey : wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Status          : Active
UserName        : David
```

- For API details, see CreateAccessKey in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_key(user_name):
    """
    Creates an access key for the specified user. Each user can have a
    maximum of two keys.

    :param user_name: The name of the user.
```

```
    :return: The created access key.
    """
    try:
        key_pair = iam.User(user_name).create_access_key_pair()
        logger.info(
            "Created access key pair for %s. Key ID is %s.",
            key_pair.user_name,
            key_pair.id,
        )
    except ClientError:
        logger.exception("Couldn't create access key pair for %s.", user_name)
        raise
    else:
        return key_pair
```

- For API details, see CreateAccessKey in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, deactivates, and deletes access keys.

```
# Manages access keys for IAM users
class AccessKeyManager
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'AccessKeyManager'
  end

  # Lists access keys for a user
  #
```

```
  # @param user_name [String] The name of the user.
  def list_access_keys(user_name)
    response = @iam_client.list_access_keys(user_name: user_name)
    if response.access_key_metadata.empty?
      @logger.info("No access keys found for user '#{user_name}'.")
    else
      response.access_key_metadata.map(&:access_key_id)
    end
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Error listing access keys: cannot find user '#{user_name}'.")
    []
  rescue StandardError => e
    @logger.error("Error listing access keys: #{e.message}")
    []
  end

  # Creates an access key for a user
  #
  # @param user_name [String] The name of the user.
  # @return [Boolean]
  def create_access_key(user_name)
    response = @iam_client.create_access_key(user_name: user_name)
    access_key = response.access_key
    @logger.info("Access key created for user '#{user_name}':
#{access_key.access_key_id}")
    access_key
  rescue Aws::IAM::Errors::LimitExceeded
    @logger.error('Error creating access key: limit exceeded. Cannot create
more.')
    nil
  rescue StandardError => e
    @logger.error("Error creating access key: #{e.message}")
    nil
  end

  # Deactivates an access key
  #
  # @param user_name [String] The name of the user.
  # @param access_key_id [String] The ID for the access key.
  # @return [Boolean]
  def deactivate_access_key(user_name, access_key_id)
    @iam_client.update_access_key(
      user_name: user_name,
      access_key_id: access_key_id,
```

```
      status: 'Inactive'
    )
    true
  rescue StandardError => e
    @logger.error("Error deactivating access key: #{e.message}")
    false
  end

  # Deletes an access key
  #
  # @param user_name [String] The name of the user.
  # @param access_key_id [String] The ID for the access key.
  # @return [Boolean]
  def delete_access_key(user_name, access_key_id)
    @iam_client.delete_access_key(
      user_name: user_name,
      access_key_id: access_key_id
    )
    true
  rescue StandardError => e
    @logger.error("Error deleting access key: #{e.message}")
    false
  end
end
```

- For API details, see CreateAccessKey in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn create_access_key(client: &iamClient, user_name: &str) ->
 Result<AccessKey, iamError> {
    let mut tries: i32 = 0;
    let max_tries: i32 = 10;
```

```
    let response: Result<CreateAccessKeyOutput, SdkError<CreateAccessKeyError>> =
 loop {
        match client.create_access_key().user_name(user_name).send().await {
            Ok(inner_response) => {
                break Ok(inner_response);
            }
            Err(e) => {
                tries += 1;
                if tries > max_tries {
                    break Err(e);
                }
                sleep(Duration::from_secs(2)).await;
            }
        }
    };

    Ok(response.unwrap().access_key.unwrap())
}
```

- For API details, see CreateAccessKey in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->createaccesskey(
          iv_username = iv_user_name ).
        MESSAGE 'Access key created successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'User does not exist.' TYPE 'E'.
      CATCH /aws1/cx_iamlimitexceededex.
        MESSAGE 'Maximum number of access keys reached.' TYPE 'E'.
    ENDTRY.
```

- For API details, see CreateAccessKey in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func createAccessKey(userName: String) async throws ->
 IAMClientTypes.AccessKey {
        let input = CreateAccessKeyInput(
            userName: userName
        )
        do {
            let output = try await iamClient.createAccessKey(input: input)
            guard let accessKey = output.accessKey else {
                throw ServiceHandlerError.keyError
            }
            return accessKey
        } catch {
            print("ERROR: createAccessKey:", dump(error))
            throw error
        }
    }
```

- For API details, see CreateAccessKey in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateAccountAlias` with an AWS SDK or CLI

The following code examples show how to use `CreateAccountAlias`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage your account

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::createAccountAlias(const Aws::String &aliasName,
                                     const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::CreateAccountAliasRequest request;
    request.SetAccountAlias(aliasName);

    Aws::IAM::Model::CreateAccountAliasOutcome outcome = iam.CreateAccountAlias(
            request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating account alias " << aliasName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully created account alias " << aliasName <<
                  std::endl;
    }
```

```
    return outcome.IsSuccess();
}
```

- For API details, see CreateAccountAlias in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To create an account alias

The following `create-account-alias` example creates the alias `examplecorp` for your AWS account.

```
aws iam create-account-alias \
    --account-alias examplecorp
```

This command produces no output.

For more information, see Your AWS account ID and its alias in the *AWS IAM User Guide*.

- For API details, see CreateAccountAlias in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.CreateAccountAliasRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateAccountAlias {
    public static void main(String[] args) {
        final String usage = """
                Usage:
                    <alias>\s

                Where:
                    alias - The account alias to create (for example,
 myawsaccount).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alias = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        createIAMAccountAlias(iam, alias);
        iam.close();
        System.out.println("Done");
    }

    public static void createIAMAccountAlias(IamClient iam, String alias) {
        try {
            CreateAccountAliasRequest request =
 CreateAccountAliasRequest.builder()
                    .accountAlias(alias)
                    .build();

            iam.createAccountAlias(request);
            System.out.println("Successfully created account alias: " + alias);
```

```
        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see CreateAccountAlias in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the account alias.

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.

- For API details, see CreateAccountAlias in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateAccountAlias in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createIAMAccountAlias(alias: String) {
    val request =
        CreateAccountAliasRequest {
            accountAlias = alias
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.createAccountAlias(request)
        println("Successfully created account alias named $alias")
    }
}
```

- For API details, see CreateAccountAlias in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example changes the account alias for your AWS account to `mycompanyaws`. The address of the user logon page chages to https://mycompanyaws.signin.aws.amazon.com/console. The original URL using your account ID number instead of the alias (https://<accountidnumber>.signin.aws.amazon.com/console) continues to work. However, any previously defined alias-based URLs stop working.

```
New-IAMAccountAlias -AccountAlias mycompanyaws
```

- For API details, see CreateAccountAlias in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example changes the account alias for your AWS account to `mycompanyaws`. The address of the user logon page chages to https://mycompanyaws.signin.aws.amazon.com/console. The original URL using your account ID number instead of the alias (https://<accountidnumber>.signin.aws.amazon.com/console) continues to work. However, any previously defined alias-based URLs stop working.

```
New-IAMAccountAlias -AccountAlias mycompanyaws
```

- For API details, see CreateAccountAlias in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_alias(alias):
    """
    Creates an alias for the current account. The alias can be used in place of
 the
    account ID in the sign-in URL. An account can have only one alias. When a new
    alias is created, it replaces any existing alias.

    :param alias: The alias to assign to the account.
    """

    try:
        iam.create_account_alias(AccountAlias=alias)
        logger.info("Created an alias '%s' for your account.", alias)
    except ClientError:
        logger.exception("Couldn't create alias '%s' for your account.", alias)
        raise
```

- For API details, see CreateAccountAlias in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List, create, and delete account aliases.

```
class IAMAliasManager
  # Initializes the IAM client and logger
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client.
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
  end

  # Lists available AWS account aliases.
  def list_aliases
    response = @iam_client.list_account_aliases

    if response.account_aliases.count.positive?
      @logger.info('Account aliases are:')
      response.account_aliases.each { |account_alias| @logger.info("
 #{account_alias}") }
    else
      @logger.info('No account aliases found.')
    end
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing account aliases: #{e.message}")
  end

  # Creates an AWS account alias.
  #
  # @param account_alias [String] The name of the account alias to create.
  # @return [Boolean] true if the account alias was created; otherwise, false.
  def create_account_alias(account_alias)
    @iam_client.create_account_alias(account_alias: account_alias)
    true
```

```
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating account alias: #{e.message}")
    false
  end

  # Deletes an AWS account alias.
  #
  # @param account_alias [String] The name of the account alias to delete.
  # @return [Boolean] true if the account alias was deleted; otherwise, false.
  def delete_account_alias(account_alias)
    @iam_client.delete_account_alias(account_alias: account_alias)
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting account alias: #{e.message}")
    false
  end
end
```

- For API details, see CreateAccountAlias in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->createaccountalias(
          iv_accountalias = iv_account_alias ).
        MESSAGE 'Account alias created successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamentityalrdyexex.
        MESSAGE 'Account alias already exists.' TYPE 'E'.
      CATCH /aws1/cx_iamlimitexceededex.
        MESSAGE 'Account alias limit exceeded.' TYPE 'E'.
    ENDTRY.
```

- For API details, see CreateAccountAlias in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateGroup` with an AWS SDK or CLI

The following code examples show how to use `CreateGroup`.

CLI

AWS CLI

To create an IAM group

The following `create-group` example creates an IAM group named `Admins`.

```
aws iam create-group \
    --group-name Admins
```

Output:

```
{
    "Group": {
        "Path": "/",
        "CreateDate": "2015-03-09T20:30:24.940Z",
        "GroupId": "AIDGPMS9RO4H3FEXAMPLE",
        "Arn": "arn:aws:iam::123456789012:group/Admins",
        "GroupName": "Admins"
    }
}
```

For more information, see Creating IAM user groups in the *AWS IAM User Guide*.

- For API details, see CreateGroup in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For API details, see CreateGroup in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new IAM group named `Developers`.

```
New-IAMGroup -GroupName Developers
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Developers
```

```
CreateDate : 4/14/2015 11:21:31 AM
GroupId    : QNEJ5PM4NFSQCEXAMPLE1
GroupName  : Developers
Path       : /
```

- For API details, see CreateGroup in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new IAM group named `Developers`.

```
New-IAMGroup -GroupName Developers
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Developers
CreateDate : 4/14/2015 11:21:31 AM
GroupId    : QNEJ5PM4NFSQCEXAMPLE1
GroupName  : Developers
Path       : /
```

- For API details, see CreateGroup in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateInstanceProfile` with an AWS SDK or CLI

The following code examples show how to use `CreateInstanceProfile`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Build and manage a resilient service
- Creating a container service for virtual machine instances
- Getting started with managed streaming
- Run CPU stress tests on virtual machine instances using fault injection

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Create a policy, role, and profile that is associated with instances with
a specified name.
   /// An instance's associated profile defines a role that is assumed by the
   /// instance.The role has attached policies that specify the AWS permissions
granted to
   /// clients that run on the instance.
   /// </summary>
   /// <param name="policyName">Name to use for the policy.</param>
   /// <param name="roleName">Name to use for the role.</param>
   /// <param name="profileName">Name to use for the profile.</param>
   /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
   /// <param name="awsManagedPolicies">AWS Managed policies to be attached to
the role.</param>
   /// <returns>The Arn of the profile.</returns>
   public async Task<string> CreateInstanceProfileWithName(
       string policyName,
       string roleName,
       string profileName,
       string ssmOnlyPolicyFile,
       List<string>? awsManagedPolicies = null)
   {

       var assumeRoleDoc = "{" +
                                   "\"Version\": \"2012-10-17\"," +
                                   "\"Statement\": [{" +
                                        "\"Effect\": \"Allow\"," +
                                        "\"Principal\": {" +
                                        "\"Service\": [" +
                                            "\"ec2.amazonaws.com\"" +
                                        "]" +
                                        "}," +
```

```
                                "\"Action\": \"sts:AssumeRole\"" +
                                "}]" +
                            "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
```

```
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await
_amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
```

```
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }
```

- For API details, see CreateInstanceProfile in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To create an instance profile

The following `create-instance-profile` example creates an instance profile named `Webserver`.

```
aws iam create-instance-profile \
    --instance-profile-name Webserver
```

Output:

```
{
    "InstanceProfile": {
        "InstanceProfileId": "AIPAJMBYC7DLSPEXAMPLE",
        "Roles": [],
        "CreateDate": "2015-03-09T20:33:19.626Z",
        "InstanceProfileName": "Webserver",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:instance-profile/Webserver"
    }
```

```
}
```

To add a role to an instance profile, use the `add-role-to-instance-profile` command.

For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *AWS IAM User Guide*.

- For API details, see CreateInstanceProfile in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

- For API details, see CreateInstanceProfile in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new IAM instance profile named `ProfileForDevEC2Instance`. You must separately run the `Add-IAMRoleToInstanceProfile` command to associate the instance profile with an existing IAM role that provides permissions to the instance. Finally, attach the instance

profile to an EC2 instance when you launch it. To do that, use the `New-EC2Instance` cmdlet with either the `InstanceProfile_Arn` or `InstanceProfile_Name` parameter.

```
New-IAMInstanceProfile -InstanceProfileName ProfileForDevEC2Instance
```

Output:

```
Arn                 : arn:aws:iam::123456789012:instance-profile/
ProfileForDevEC2Instance
CreateDate          : 4/14/2015 11:31:39 AM
InstanceProfileId   : DYMFXL556EY46EXAMPLE1
InstanceProfileName : ProfileForDevEC2Instance
Path                : /
Roles               : {}
```

- For API details, see CreateInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new IAM instance profile named `ProfileForDevEC2Instance`. You must separately run the `Add-IAMRoleToInstanceProfile` command to associate the instance profile with an existing IAM role that provides permissions to the instance. Finally, attach the instance profile to an EC2 instance when you launch it. To do that, use the `New-EC2Instance` cmdlet with either the `InstanceProfile_Arn` or `InstanceProfile_Name` parameter.

```
New-IAMInstanceProfile -InstanceProfileName ProfileForDevEC2Instance
```

Output:

```
Arn                 : arn:aws:iam::123456789012:instance-profile/
ProfileForDevEC2Instance
CreateDate          : 4/14/2015 11:31:39 AM
InstanceProfileId   : DYMFXL556EY46EXAMPLE1
InstanceProfileName : ProfileForDevEC2Instance
Path                : /
Roles               : {}
```

- For API details, see CreateInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example creates a policy, role, and instance profile and links them all together.

```
class AutoScalingWrapper:
    """
    Encapsulates Amazon EC2 Auto Scaling and EC2 management actions.
    """

    def __init__(
        self,
        resource_prefix: str,
        inst_type: str,
        ami_param: str,
        autoscaling_client: boto3.client,
        ec2_client: boto3.client,
        ssm_client: boto3.client,
        iam_client: boto3.client,
    ):
        """
        Initializes the AutoScaler class with the necessary parameters.

        :param resource_prefix: The prefix for naming AWS resources that are
 created by this class.
        :param inst_type: The type of EC2 instance to create, such as t3.micro.
        :param ami_param: The Systems Manager parameter used to look up the AMI
 that is created.
        :param autoscaling_client: A Boto3 EC2 Auto Scaling client.
        :param ec2_client: A Boto3 EC2 client.
        :param ssm_client: A Boto3 Systems Manager client.
        :param iam_client: A Boto3 IAM client.
        """
        self.inst_type = inst_type
        self.ami_param = ami_param
        self.autoscaling_client = autoscaling_client
```

```
        self.ec2_client = ec2_client
        self.ssm_client = ssm_client
        self.iam_client = iam_client
        sts_client = boto3.client("sts")
        self.account_id = sts_client.get_caller_identity()["Account"]

        self.key_pair_name = f"{resource_prefix}-key-pair"
        self.launch_template_name = f"{resource_prefix}-template-"
        self.group_name = f"{resource_prefix}-group"

        # Happy path
        self.instance_policy_name = f"{resource_prefix}-pol"
        self.instance_role_name = f"{resource_prefix}-role"
        self.instance_profile_name = f"{resource_prefix}-prof"

        # Failure mode
        self.bad_creds_policy_name = f"{resource_prefix}-bc-pol"
        self.bad_creds_role_name = f"{resource_prefix}-bc-role"
        self.bad_creds_profile_name = f"{resource_prefix}-bc-prof"


    def create_instance_profile(
        self,
        policy_file: str,
        policy_name: str,
        role_name: str,
        profile_name: str,
        aws_managed_policies: Tuple[str, ...] = (),
    ) -> str:
        """
        Creates a policy, role, and profile that is associated with instances
created by
        this class. An instance's associated profile defines a role that is
assumed by the
        instance. The role has attached policies that specify the AWS permissions
granted to
        clients that run on the instance.

        :param policy_file: The name of a JSON file that contains the policy
definition to
                            create and attach to the role.
        :param policy_name: The name to give the created policy.
        :param role_name: The name to give the created role.
        :param profile_name: The name to the created profile.
```

```
        :param aws_managed_policies: Additional AWS-managed policies that are
attached to
                                     the role, such as
AmazonSSMManagedInstanceCore to grant
                                     use of Systems Manager to send commands to
the instance.
        :return: The ARN of the profile that is created.
        """
        assume_role_doc = {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {"Service": "ec2.amazonaws.com"},
                    "Action": "sts:AssumeRole",
                }
            ],
        }
        policy_arn = self.create_policy(policy_file, policy_name)
        self.create_role(role_name, assume_role_doc)
        self.attach_policy(role_name, policy_arn, aws_managed_policies)

        try:
            profile_response = self.iam_client.create_instance_profile(
                InstanceProfileName=profile_name
            )
            waiter = self.iam_client.get_waiter("instance_profile_exists")
            waiter.wait(InstanceProfileName=profile_name)
            time.sleep(10)  # wait a little longer
            profile_arn = profile_response["InstanceProfile"]["Arn"]
            self.iam_client.add_role_to_instance_profile(
                InstanceProfileName=profile_name, RoleName=role_name
            )
            log.info("Created profile %s and added role %s.", profile_name,
role_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityAlreadyExists":
                prof_response = self.iam_client.get_instance_profile(
                    InstanceProfileName=profile_name
                )
                profile_arn = prof_response["InstanceProfile"]["Arn"]
                log.info(
                    "Instance profile %s already exists, nothing to do.",
profile_name
```

```
                )
            log.error(f"Full error:\n\t{err}")
        return profile_arn
```

- For API details, see CreateInstanceProfile in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateLoginProfile` with a CLI

The following code examples show how to use `CreateLoginProfile`.

CLI

AWS CLI

To create a password for an IAM user

The following `create-login-profile` example creates a password for an IAM user. To create the password, we recommend using the `--cli-input-json` parameter to pass a JSON file that contains the password. Using this method, you can create a strong password with non-alphanumeric characters. It can be difficult to create a password with non-alphanumeric characters when you pass it as a command line parameter.

To use the `--cli-input-json` parameter, start by using the `create-login-profile` command with the `--generate-cli-skeleton` parameter, as in the following example.

```
aws iam create-login-profile \
    --generate-cli-skeleton > create-login-profile.json
```

The previous command creates a JSON file called create-login-profile.json that you can use to fill in the information for a subsequent `create-login-profile` command.

Next, to create a password for an IAM user, use the `create-login-profile` command again, this time passing the `--cli-input-json` parameter to specify your JSON file. The following `create-login-profile` command uses the `--cli-input-json` parameter with a JSON file called create-login-profile.json.

```
aws iam create-login-profile \
    --cli-input-json file://create-login-profile.json
```

Contents of `create-login-profile.json`:

```
{
    "UserName": "Bob",
    "Password": "&1-3a6u:RA0djs",
    "PasswordResetRequired": true
}
```

Output:

```
{
    "LoginProfile": {
        "UserName": "Bob",
        "CreateDate": "2015-03-10T20:55:40.274Z",
        "PasswordResetRequired": true
    }
}
```

If the new password violates the account password policy, the command returns a `PasswordPolicyViolation` error.

To change the password for a user that already has one, use `update-login-profile`. To set a password policy for the account, use the `update-account-password-policy` command.

If the account password policy allows them to, IAM users can change their own passwords using the `change-password` command.

For more information, see Managing passwords for IAM users in the *AWS IAM User Guide*.

- For API details, see CreateLoginProfile in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a (temporary) password for the IAM user named Bob, and sets the flag that requires the user to change the password the next time Bob signs in.

```
New-IAMLoginProfile -UserName Bob -Password P@ssw0rd -PasswordResetRequired $true
```

Output:

```
CreateDate                  PasswordResetRequired             UserName
----------                  ---------------------             --------
4/14/2015 12:26:30 PM       True                              Bob
```

- For API details, see CreateLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a (temporary) password for the IAM user named Bob, and sets the flag that requires the user to change the password the next time Bob signs in.

```
New-IAMLoginProfile -UserName Bob -Password P@ssw0rd -PasswordResetRequired $true
```

Output:

```
CreateDate                  PasswordResetRequired             UserName
----------                  ---------------------             --------
4/14/2015 12:26:30 PM       True                              Bob
```

- For API details, see CreateLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateOpenIdConnectProvider` with a CLI

The following code examples show how to use `CreateOpenIdConnectProvider`.

CLI

AWS CLI

To create an OpenID Connect (OIDC) provider

The following `create-open-id-connect-provider` example creates an OpenID Connect (OIDC) provider. To create the provider, we recommend using the `--cli-input-json` parameter to pass a JSON file that contains the required parameters. When you create an OIDC provider, you must pass the URL of the provider, and the URL must begin with `https://`. It can be difficult to pass the URL as a command line parameter, because the colon (:) and forward slash (/) characters have special meaning in some command line environments. Using the `--cli-input-json` parameter gets around this limitation.

To use the `--cli-input-json` parameter, start by using the `create-open-id-connect-provider` command with the `--generate-cli-skeleton` parameter, as in the following example.

```
aws iam create-open-id-connect-provider \
    --generate-cli-skeleton > create-open-id-connect-provider.json
```

The previous command creates a JSON file called create-open-id-connect-provider.json that you can use to fill in the information for a subsequent `create-open-id-connect-provider` command.

Next, to create the OpenID Connect (OIDC) provider, use the `create-open-id-connect-provider` command again, this time passing the `--cli-input-json` parameter to specify your JSON file. The following `create-open-id-connect-provider` command uses the `--cli-input-json` parameter with a JSON file called create-open-id-connect-provider.json.

```
aws iam create-open-id-connect-provider \
    --cli-input-json file://create-open-id-connect-provider.json
```

Contents of `create-open-id-connect-provider.json`:

```
{
    "Url": "https://server.example.com",
    "ClientIDList": [
        "example-application-ID"
```

```
    ],
    "ThumbprintList": [
        "c3768084dfb3d2b68b7897bf5f565da8eEXAMPLE"
    ]
}
```

If your OIDC provider's discovery and JWKS endpoints use different certificates, include thumbprints for both endpoints in the `ThumbprintList`.

Output:

```
{
    "OpenIDConnectProviderArn": "arn:aws:iam::123456789012:oidc-provider/
server.example.com"
}
```

For more information about OIDC providers, see Creating OpenID Connect (OIDC) identity providers in the *AWS IAM User Guide*.

For more information about obtaining thumbprints for an OIDC provider, see Obtaining the thumbprint for an OpenID Connect Identity Provider in the *AWS IAM User Guide*.

- For API details, see CreateOpenIdConnectProvider in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates an IAM OIDC provider associated with the OIDC compatible provider service found at the URL `https://example.oidcprovider.com` and the client ID `my-testapp-1`. The OIDC provider supplies the thumbprint. To authenticate the thumbprint, follow the steps at http://docs.aws.amazon.com/IAM/latest/UserGuide/identity-providers-oidc-obtain-thumbprint.html.

```
New-IAMOpenIDConnectProvider -Url https://example.oidcprovider.com -ClientIDList
 my-testapp-1 -ThumbprintList 990F419EXAMPLEECF12DDEDA5EXAMPLE52F20D9E
```

Output:

```
arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com
```

- For API details, see CreateOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates an IAM OIDC provider associated with the OIDC compatible provider service found at the URL `https://example.oidcprovider.com` and the client ID `my-testapp-1`. The OIDC provider supplies the thumbprint. To authenticate the thumbprint, follow the steps at http://docs.aws.amazon.com/IAM/latest/UserGuide/identity-providers-oidc-obtain-thumbprint.html.

```
New-IAMOpenIDConnectProvider -Url https://example.oidcprovider.com -ClientIDList
 my-testapp-1 -ThumbprintList 990F419EXAMPLEECF12DDEDA5EXAMPLE52F20D9E
```

Output:

```
arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com
```

- For API details, see CreateOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreatePolicy` with an AWS SDK or CLI

The following code examples show how to use `CreatePolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Create read-only and read-write users
- Creating a managed monitoring workspace
- Getting started with managed streaming
- Getting started with workflow orchestration

- Manage policies
- Set up Attribute-Based Access Control
- Setting up systems management
- Work with the IAM Policy Builder API

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</
param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
 policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new
 CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }
```

- For API details, see CreatePolicy in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function iam_create_policy
#
# This function creates an IAM policy.
#
# Parameters:
#       -n policy_name -- The name of the IAM policy.
#       -p policy_json -- The policy document.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function iam_create_policy() {
  local policy_name policy_document response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_policy"
    echo "Creates an AWS Identity and Access Management (IAM) policy."
    echo "  -n policy_name   The name of the IAM policy."
    echo "  -p policy_json -- The policy document."
```

```
  echo ""
}

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
  case "${option}" in
    n) policy_name="${OPTARG}" ;;
    p) policy_document="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$policy_name" ]]; then
  errecho "ERROR: You must provide a policy name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$policy_document" ]]; then
  errecho "ERROR: You must provide a policy document with the -p parameter."
  usage
  return 1
fi

response=$(aws iam create-policy \
  --policy-name "$policy_name" \
  --policy-document "$policy_document" \
  --output text \
  --query Policy.Arn)

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports create-policy operation failed.\n$response"
```

```
    return 1
  fi

  echo "$response"
}
```

- For API details, see CreatePolicy in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
Aws::String AwsDoc::IAM::createPolicy(const Aws::String &policyName,
                                      const Aws::String &rsrcArn,
                                      const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::CreatePolicyRequest request;
    request.SetPolicyName(policyName);
    request.SetPolicyDocument(BuildSamplePolicyDocument(rsrcArn));

    Aws::IAM::Model::CreatePolicyOutcome outcome = iam.CreatePolicy(request);
    Aws::String result;
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating policy " << policyName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        result = outcome.GetResult().GetPolicy().GetArn();
        std::cout << "Successfully created policy " << policyName <<
                  std::endl;
    }

    return result;
```

```
}

Aws::String AwsDoc::IAM::BuildSamplePolicyDocument(const Aws::String &rsrc_arn) {
    std::stringstream stringStream;
    stringStream << "{"
                 << "  \"Version\": \"2012-10-17\","
                 << "  \"Statement\": ["
                 << "    {"
                 << "        \"Effect\": \"Allow\","
                 << "        \"Action\": \"logs:CreateLogGroup\","
                 << "        \"Resource\": \""
                 << rsrc_arn
                 << "\""
                 << "    },"
                 << "    {"
                 << "        \"Effect\": \"Allow\","
                 << "        \"Action\": ["
                 << "            \"dynamodb:DeleteItem\","
                 << "            \"dynamodb:GetItem\","
                 << "            \"dynamodb:PutItem\","
                 << "            \"dynamodb:Scan\","
                 << "            \"dynamodb:UpdateItem\""
                 << "       ],"
                 << "       \"Resource\": \""
                 << rsrc_arn
                 << "\""
                 << "    }"
                 << "   ]"
                 << "}";

    return stringStream.str();
}
```

- For API details, see CreatePolicy in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To create a customer managed policy

The following `create-policy` example creates a customer managed policy named `my-policy`. The file `policy.json` is a JSON document in the current folder that grants read only access to the `shared` folder in an Amazon S3 bucket named `amzn-s3-demo-bucket`.

```
aws iam create-policy \
    --policy-name my-policy \
    --policy-document file://policy.json
```

Contents of `policy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/shared/*"
            ]
        }
    ]
}
```

Output:

```
{
    "Policy": {
        "PolicyName": "my-policy",
        "CreateDate": "2015-06-01T19:31:18.620Z",
        "AttachmentCount": 0,
        "IsAttachable": true,
        "PolicyId": "ZXR6A36LTYANPAI7NJ5UV",
        "DefaultVersionId": "v1",
        "Path": "/",
        "Arn": "arn:aws:iam::0123456789012:policy/my-policy",
        "UpdateDate": "2015-06-01T19:31:18.620Z"
    }
}
```

For more information on using files as input for string parameters, see Specify parameter values for the AWS CLI in the *AWS CLI User Guide*.

Example 2: To create a customer managed policy with a description

The following `create-policy` example creates a customer managed policy named `my-policy` with an immutable description.

The file `policy.json` is a JSON document in the current folder that grants access to all Put, List, and Get actions for an Amazon S3 bucket named `amzn-s3-demo-bucket`.

```
aws iam create-policy \
    --policy-name my-policy \
    --policy-document file://policy.json \
    --description "This policy grants access to all Put, Get, and List actions
 for amzn-s3-demo-bucket"
```

Contents of `policy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket*",
                "s3:PutBucket*",
                "s3:GetBucket*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

Output:

```
{
    "Policy": {
        "PolicyName": "my-policy",
```

```
        "PolicyId": "ANPAWGSUGIDPEXAMPLE",
        "Arn": "arn:aws:iam::123456789012:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2023-05-24T22:38:47+00:00",
        "UpdateDate": "2023-05-24T22:38:47+00:00"
    }
}
```

For more information on Idenity-based Policies, see Identity-based policies and resource-based policies in the *AWS IAM User Guide*.

Example 3: To create a customer managed policy with tags

The following `create-policy` example creates a customer managed policy named `my-policy` with tags. This example uses the `--tags` parameter with the following JSON-formatted tags: `'{"Key": "Department", "Value": "Accounting"}' '{"Key": "Location", "Value": "Seattle"}'`. Alternatively, the `--tags` parameter can be used with tags in the shorthand format: `'Key=Department,Value=Accounting Key=Location,Value=Seattle'`.

The file `policy.json` is a JSON document in the current folder that grants access to all Put, List, and Get actions for an Amazon S3 bucket named `amzn-s3-demo-bucket`.

```
aws iam create-policy \
    --policy-name my-policy \
    --policy-document file://policy.json \
    --tags '{"Key": "Department", "Value": "Accounting"}' '{"Key": "Location",
 "Value": "Seattle"}'
```

Contents of `policy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "s3:ListBucket*",
                "s3:PutBucket*",
                "s3:GetBucket*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

Output:

```
{
    "Policy": {
        "PolicyName": "my-policy",
        "PolicyId": "ANPAWGSUGIDPEXAMPLE",
        "Arn": "arn:aws:iam::12345678012:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2023-05-24T23:16:39+00:00",
        "UpdateDate": "2023-05-24T23:16:39+00:00",
        "Tags": [
            {
                "Key": "Department",
                "Value": "Accounting"
            },
            {
                "Key": "Location",
                "Value": "Seattle"
            }
        ]
    }
}
```

For more information on Tagging policies, see Tagging customer managed policies in the *AWS IAM User Guide*.

- For API details, see CreatePolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    []string
```

```
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
}

// CreatePolicy creates a policy that grants a list of actions to the specified
 resource.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(ctx context.Context, policyName string,
 actions []string,
 resourceArn string) (*types.Policy, error) {
 var policy *types.Policy
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(resourceArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
 resourceArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreatePolicy(ctx, &iam.CreatePolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
 })
 if err != nil {
  log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}
```

- For API details, see CreatePolicy in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.iam.model.CreatePolicyRequest;
import software.amazon.awssdk.services.iam.model.CreatePolicyResponse;
import software.amazon.awssdk.services.iam.model.GetPolicyRequest;
import software.amazon.awssdk.services.iam.model.GetPolicyResponse;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.waiters.IamWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreatePolicy {

    public static final String PolicyDocument = "{" +
            "  \"Version\": \"2012-10-17\"," +
            "  \"Statement\": [" +
            "    {" +
            "        \"Effect\": \"Allow\"," +
            "        \"Action\": [" +
            "            \"dynamodb:DeleteItem\"," +
            "            \"dynamodb:GetItem\"," +
            "            \"dynamodb:PutItem\"," +
            "            \"dynamodb:Scan\"," +
            "            \"dynamodb:UpdateItem\"" +
```

```
            "       ]," +
            "       \"Resource\": \"*\"" +
            "    }" +
            "   ]" +
            "}";

    public static void main(String[] args) {

        final String usage = """
                Usage:
                    CreatePolicy <policyName>\s

                Where:
                    policyName - A unique policy name.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String policyName = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        String result = createIAMPolicy(iam, policyName);
        System.out.println("Successfully created a policy with this ARN value: "
+ result);
        iam.close();
    }

    public static String createIAMPolicy(IamClient iam, String policyName) {
        try {
            // Create an IamWaiter object.
            IamWaiter iamWaiter = iam.waiter();

            CreatePolicyRequest request = CreatePolicyRequest.builder()
                    .policyName(policyName)
                    .policyDocument(PolicyDocument)
                    .build();

            CreatePolicyResponse response = iam.createPolicy(request);
```

```
            // Wait until the policy is created.
            GetPolicyRequest polRequest = GetPolicyRequest.builder()
                    .policyArn(response.policy().arn())
                    .build();

            WaiterResponse<GetPolicyResponse> waitUntilPolicyExists =
 iamWaiter.waitUntilPolicyExists(polRequest);

 waitUntilPolicyExists.matched().response().ifPresent(System.out::println);
            return response.policy().arn();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see CreatePolicy in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the policy.

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
```

```
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: "*",
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreatePolicy in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
```

```
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreatePolicy in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createIAMPolicy(policyNameVal: String?): String {
    val policyDocumentVal = """
    {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "dynamodb:DeleteItem",
                    "dynamodb:GetItem",
                    "dynamodb:PutItem",
                    "dynamodb:Scan",
                    "dynamodb:UpdateItem"
                ],
                "Resource": "*"
            }
        ]
    }
    """.trimIndent()

    val request =
        CreatePolicyRequest {
            policyName = policyNameVal
            policyDocument = policyDocumentVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createPolicy(request)
        return response.policy?.arn.toString()
    }
}
```

- For API details, see CreatePolicy in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$listAllBucketsPolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Action\": \"s3:ListAllMyBuckets\",
                    \"Resource\": \"arn:aws:s3:::*\"}]
}";
$listAllBucketsPolicy = $service->createPolicy("iam_demo_policy_$uuid",
 $listAllBucketsPolicyDocument);
echo "Created policy: {$listAllBucketsPolicy['PolicyName']}\n";

    /**
     * @param string $policyName
     * @param string $policyDocument
     * @return array
     */
    public function createPolicy(string $policyName, string $policyDocument)
    {
        $result = $this->customWaiter(function () use ($policyName,
 $policyDocument) {
            return $this->iamClient->createPolicy([
                'PolicyName' => $policyName,
                'PolicyDocument' => $policyDocument,
            ]);
        });
        return $result['Policy'];
    }
```

- For API details, see CreatePolicy in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new IAM policy in the current AWS account named `MySamplePolicy` The file `MySamplePolicy.json` provides the policy content. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file.

```
New-IAMPolicy -PolicyName MySamplePolicy -PolicyDocument (Get-Content -Raw
 MySamplePolicy.json)
```

Output:

```
Arn              : arn:aws:iam::123456789012:policy/MySamplePolicy
AttachmentCount  : 0
CreateDate       : 4/14/2015 2:45:59 PM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : LD4KP6HVFE7WGEXAMPLE1
PolicyName       : MySamplePolicy
UpdateDate       : 4/14/2015 2:45:59 PM
```

- For API details, see CreatePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new IAM policy in the current AWS account named `MySamplePolicy` The file `MySamplePolicy.json` provides the policy content. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file.

```
New-IAMPolicy -PolicyName MySamplePolicy -PolicyDocument (Get-Content -Raw
 MySamplePolicy.json)
```

Output:

```
Arn              : arn:aws:iam::123456789012:policy/MySamplePolicy
AttachmentCount  : 0
CreateDate       : 4/14/2015 2:45:59 PM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : LD4KP6HVFE7WGEXAMPLE1
PolicyName       : MySamplePolicy
UpdateDate       : 4/14/2015 2:45:59 PM
```

- For API details, see CreatePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_policy(name, description, actions, resource_arn):
    """
    Creates a policy that contains a single statement.

    :param name: The name of the policy to create.
    :param description: The description of the policy.
    :param actions: The actions allowed by the policy. These typically take the
                    form of service:action, such as s3:PutObject.
    :param resource_arn: The Amazon Resource Name (ARN) of the resource this
 policy
                         applies to. This ARN can contain wildcards, such as
                         'arn:aws:s3:::amzn-s3-demo-bucket/*' to allow actions on
 all objects
                         in the bucket named 'amzn-s3-demo-bucket'.
    :return: The newly created policy.
    """
    policy_doc = {
        "Version":"2012-10-17",
```

```
        "Statement": [{"Effect": "Allow", "Action": actions, "Resource":
 resource_arn}],
    }
    try:
        policy = iam.create_policy(
            PolicyName=name,
            Description=description,
            PolicyDocument=json.dumps(policy_doc),
        )
        logger.info("Created policy %s.", policy.arn)
    except ClientError:
        logger.exception("Couldn't create policy %s.", name)
        raise
    else:
        return policy
```

- For API details, see CreatePolicy in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, attaches, and detaches role policies.

```
# Manages policies in AWS Identity and Access Management (IAM)
class RolePolicyManager
  # Initialize with an AWS IAM client
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'PolicyManager'
```

```
  end

  # Creates a policy
  #
  # @param policy_name [String] The name of the policy
  # @param policy_document [Hash] The policy document
  # @return [String] The policy ARN if successful, otherwise nil
  def create_policy(policy_name, policy_document)
    response = @iam_client.create_policy(
      policy_name: policy_name,
      policy_document: policy_document.to_json
    )
    response.policy.arn
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating policy: #{e.message}")
    nil
  end

  # Fetches an IAM policy by its ARN
  # @param policy_arn [String] the ARN of the IAM policy to retrieve
  # @return [Aws::IAM::Types::GetPolicyResponse] the policy object if found
  def get_policy(policy_arn)
    response = @iam_client.get_policy(policy_arn: policy_arn)
    policy = response.policy
    @logger.info("Got policy '#{policy.policy_name}'. Its ID is:
#{policy.policy_id}.")
    policy
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Couldn't get policy '#{policy_arn}'. The policy does not
exist.")
    raise
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't get policy '#{policy_arn}'. Here's why: #{e.code}:
#{e.message}")
    raise
  end

  # Attaches a policy to a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def attach_policy_to_role(role_name, policy_arn)
    @iam_client.attach_role_policy(
```

```
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error attaching policy to role: #{e.message}")
    false
  end

  # Lists policy ARNs attached to a role
  #
  # @param role_name [String] The name of the role
  # @return [Array<String>] List of policy ARNs
  def list_attached_policy_arns(role_name)
    response = @iam_client.list_attached_role_policies(role_name: role_name)
    response.attached_policies.map(&:policy_arn)
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing policies attached to role: #{e.message}")
    []
  end

  # Detaches a policy from a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def detach_policy_from_role(role_name, policy_arn)
    @iam_client.detach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error detaching policy from role: #{e.message}")
    false
  end
end
```

- For API details, see CreatePolicy in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn create_policy(
    client: &iamClient,
    policy_name: &str,
    policy_document: &str,
) -> Result<Policy, iamError> {
    let policy = client
        .create_policy()
        .policy_name(policy_name)
        .policy_document(policy_document)
        .send()
        .await?;
    Ok(policy.policy.unwrap())
}
```

- For API details, see CreatePolicy in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->createpolicy(
          iv_policyname = iv_policy_name
```

```
          iv_policydocument = iv_policy_document
          iv_description = iv_description ).
        MESSAGE 'Policy created successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamentityalrdyexex.
        MESSAGE 'Policy already exists.' TYPE 'E'.
      CATCH /aws1/cx_iammalformedplydocex.
        MESSAGE 'Policy document is malformed.' TYPE 'E'.
      CATCH /aws1/cx_iamlimitexceededex.
        MESSAGE 'Policy limit exceeded.' TYPE 'E'.
    ENDTRY.
```

- For API details, see CreatePolicy in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func createPolicy(name: String, policyDocument: String) async throws -
> IAMClientTypes.Policy {
        let input = CreatePolicyInput(
            policyDocument: policyDocument,
            policyName: name
        )
        do {
            let output = try await iamClient.createPolicy(input: input)
            guard let policy = output.policy else {
                throw ServiceHandlerError.noSuchPolicy
            }
            return policy
        } catch {
            print("ERROR: createPolicy:", dump(error))
```

```
            throw error
        }
    }
```

- For API details, see CreatePolicy in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreatePolicyVersion` with an AWS SDK or CLI

The following code examples show how to use `CreatePolicyVersion`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage policies

CLI

AWS CLI

To create a new version of a managed policy

The following `create-policy-version` example creates a new v2 version of the IAM policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` and makes it the default version.

```
aws iam create-policy-version \
    --policy-arn arn:aws:iam::123456789012:policy/MyPolicy \
    --policy-document file://NewPolicyVersion.json \
    --set-as-default
```

Contents of `NewPolicyVersion.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

Output:

```
{
    "PolicyVersion": {
        "CreateDate": "2015-06-16T18:56:03.721Z",
        "VersionId": "v2",
        "IsDefaultVersion": true
    }
}
```

For more information, see Versioning IAM policies in the *AWS IAM User Guide*.

- For API details, see CreatePolicyVersion in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new "v2" version of the IAM policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` and makes it the default version. The `NewPolicyVersion.json` file provides the policy content. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file.

```
New-IAMPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/MyPolicy -
PolicyDocument (Get-content -Raw NewPolicyVersion.json) -SetAsDefault $true
```

Output:

```
CreateDate                              Document                IsDefaultVersion
           VersionId
----------                              --------                ---------------
           ---------
4/15/2015 10:54:54 AM                                           True
           v2
```

- For API details, see CreatePolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new "v2" version of the IAM policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` and makes it the default version. The `NewPolicyVersion.json` file provides the policy content. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file.

```
New-IAMPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/MyPolicy -
PolicyDocument (Get-content -Raw NewPolicyVersion.json) -SetAsDefault $true
```

Output:

```
CreateDate                              Document                IsDefaultVersion
           VersionId
----------                              --------                ---------------
           ---------
4/15/2015 10:54:54 AM                                           True
           v2
```

- For API details, see CreatePolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_policy_version(policy_arn, actions, resource_arn, set_as_default):
    """
    Creates a policy version. Policies can have up to five versions. The default
    version is the one that is used for all resources that reference the policy.

    :param policy_arn: The ARN of the policy.
    :param actions: The actions to allow in the policy version.
    :param resource_arn: The ARN of the resource this policy version applies to.
    :param set_as_default: When True, this policy version is set as the default
                           version for the policy. Otherwise, the default
                           is not changed.
    :return: The newly created policy version.
    """
    policy_doc = {
        "Version":"2012-10-17",
        "Statement": [{"Effect": "Allow", "Action": actions, "Resource":
 resource_arn}],
    }
    try:
        policy = iam.Policy(policy_arn)
        policy_version = policy.create_version(
            PolicyDocument=json.dumps(policy_doc), SetAsDefault=set_as_default
        )
        logger.info(
            "Created policy version %s for policy %s.",
            policy_version.version_id,
            policy_version.arn,
        )
    except ClientError:
        logger.exception("Couldn't create a policy version for %s.", policy_arn)
        raise
    else:
        return policy_version
```

- For API details, see CreatePolicyVersion in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->createpolicyversion(
      iv_policyarn = iv_policy_arn
      iv_policydocument = iv_policy_document
      iv_setasdefault = iv_set_as_default ).
    MESSAGE 'Policy version created successfully.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'Policy does not exist.' TYPE 'E'.
  CATCH /aws1/cx_iammalformedplydocex.
    MESSAGE 'Policy document is malformed.' TYPE 'E'.
  CATCH /aws1/cx_iamlimitexceededex.
    MESSAGE 'Policy version limit exceeded.' TYPE 'E'.
ENDTRY.
```

- For API details, see CreatePolicyVersion in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateRole` with an AWS SDK or CLI

The following code examples show how to use `CreateRole`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Configure container service connectivity

- Create a container task for the serverless launch type
- Create a rest API with function proxy integration
- Creating a container service for virtual machine instances
- Creating a managed monitoring workspace
- Creating a monitoring dashboard with function name as a variable
- Creating your first serverless function
- Get started with serverless data warehouses
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with provisioned data warehouse clusters
- Getting started with workflow orchestration
- Manage roles
- Moving hardcoded secrets to secure secret storage
- Run CPU stress tests on virtual machine instances using fault injection
- Setting up systems management
- Using property variables in monitoring dashboards to monitor multiple serverless functions
- Work with Streams and Time-to-Live

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Create a new IAM role.
```

```
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
 rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }
```

- For API details, see CreateRole in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
```

```
# function iam_create_role
#
# This function creates an IAM role.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_json -- The assume role policy document.
#
# Returns:
#       The ARN of the role.
#     And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_create_role() {
  local role_name policy_document response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user_access_key"
    echo "Creates an AWS Identity and Access Management (IAM) role."
    echo "  -n role_name   The name of the IAM role."
    echo "  -p policy_json -- The assume role policy document."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:p:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      p) policy_document="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1
```

```
  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$policy_document" ]]; then
    errecho "ERROR: You must provide a policy document with the -p parameter."
    usage
    return 1
  fi

  response=$(aws iam create-role \
    --role-name "$role_name" \
    --assume-role-policy-document "$policy_document" \
    --output text \
    --query Role.Arn)

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- For API details, see CreateRole in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::createIamRole(
        const Aws::String &roleName,
        const Aws::String &policy,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::IAM::IAMClient client(clientConfig);
    Aws::IAM::Model::CreateRoleRequest request;

    request.SetRoleName(roleName);
    request.SetAssumeRolePolicyDocument(policy);

    Aws::IAM::Model::CreateRoleOutcome outcome = client.CreateRole(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error creating role. " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        const Aws::IAM::Model::Role iamRole = outcome.GetResult().GetRole();
        std::cout << "Created role " << iamRole.GetRoleName() << "\n";
        std::cout << "ID: " << iamRole.GetRoleId() << "\n";
        std::cout << "ARN: " << iamRole.GetArn() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see CreateRole in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To create an IAM role

The following `create-role` example creates a role named `Test-Role` and attaches a trust policy to it.

```
aws iam create-role \
    --role-name Test-Role \
    --assume-role-policy-document file://Test-Role-Trust-Policy.json
```

Contents of `Test-Role-Trust-Policy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Output:

```
{
    "Role": {
        "AssumeRolePolicyDocument": "<URL-encoded-JSON>",
        "RoleId": "AKIAIOSFODNN7EXAMPLE",
        "CreateDate": "2013-06-07T20:43:32.821Z",
        "RoleName": "Test-Role",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:role/Test-Role"
    }
}
```

The trust policy is defined as a JSON document in the *Test-Role-Trust-Policy.json* file. (The file name and extension do not have significance.) The trust policy must specify a principal.

To attach a permissions policy to a role, use the `put-role-policy` command.

For more information, see Creating IAM roles in the *AWS IAM User Guide*.

Example 2: To create an IAM role with specified maximum session duration

The following `create-role` example creates a role named `Test-Role` and sets a maximum session duration of 7200 seconds (2 hours).

```
aws iam create-role \
    --role-name Test-Role \
    --assume-role-policy-document file://Test-Role-Trust-Policy.json \
    --max-session-duration 7200
```

Contents of `Test-Role-Trust-Policy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Output:

```
{
    "Role": {
        "Path": "/",
        "RoleName": "Test-Role",
        "RoleId": "AKIAIOSFODNN7EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:role/Test-Role",
        "CreateDate": "2023-05-24T23:50:25+00:00",
        "AssumeRolePolicyDocument": {
```

```
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Sid": "Statement1",
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::123456789012:root"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

For more information, see Modifying a role maximum session duration (AWS API) in the *AWS IAM User Guide*.

Example 3: To create an IAM Role with tags

The following `create-role` example creates an IAM role `Test-Role` with tags. This example uses the `--tags` parameter flag with the following JSON-formatted tags: `'{"Key": "Department", "Value": "Accounting"}' '{"Key": "Location", "Value": "Seattle"}'`. Alternatively, the `--tags` flag can be used with tags in the shorthand format: `'Key=Department,Value=Accounting Key=Location,Value=Seattle'`.

```
aws iam create-role \
    --role-name Test-Role \
    --assume-role-policy-document file://Test-Role-Trust-Policy.json \
    --tags '{"Key": "Department", "Value": "Accounting"}' '{"Key": "Location",
 "Value": "Seattle"}'
```

Contents of `Test-Role-Trust-Policy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Output:

```
{
    "Role": {
        "Path": "/",
        "RoleName": "Test-Role",
        "RoleId": "AKIAIOSFODNN7EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:role/Test-Role",
        "CreateDate": "2023-05-25T23:29:41+00:00",
        "AssumeRolePolicyDocument": {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Sid": "Statement1",
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::123456789012:root"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Tags": [
            {
                "Key": "Department",
                "Value": "Accounting"
            },
            {
                "Key": "Location",
                "Value": "Seattle"
            }
        ]
    }
}
```

For more information, see Tagging IAM roles in the *AWS IAM User Guide*.

- For API details, see CreateRole in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// CreateRole creates a role that trusts a specified user. The trusted user can
 assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(ctx context.Context, roleName string,
 trustedUserArn string) (*types.Role, error) {
 var role *types.Role
 trustPolicy := PolicyDocument{
```

```
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Principal: map[string]string{"AWS": trustedUserArn},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyBytes, err := json.Marshal(trustPolicy)
 if err != nil {
  log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
 trustedUserArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreateRole(ctx, &iam.CreateRoleInput{
  AssumeRolePolicyDocument: aws.String(string(policyBytes)),
  RoleName:                 aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

- For API details, see CreateRole in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import software.amazon.awssdk.services.iam.model.CreateRoleRequest;
```

```
import software.amazon.awssdk.services.iam.model.CreateRoleResponse;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import java.io.FileReader;

/*
*   This example requires a trust policy document. For more information, see:
*   https://aws.amazon.com/blogs/security/how-to-use-trust-policies-with-iam-
roles/
*
*
*  In addition, set up your development environment, including your credentials.
*
*  For information, see this documentation topic:
*
*  https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class CreateRole {
    public static void main(String[] args) throws Exception {
        final String usage = """
                Usage:
                    <rolename> <fileLocation>\s

                Where:
                    rolename - The name of the role to create.\s
                    fileLocation - The location of the JSON document that
 represents the trust policy.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String rolename = args[0];
        String fileLocation = args[1];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();
```

```
        String result = createIAMRole(iam, rolename, fileLocation);
        System.out.println("Successfully created user: " + result);
        iam.close();
    }

    public static String createIAMRole(IamClient iam, String rolename, String
 fileLocation) throws Exception {
        try {
            JSONObject jsonObject = (JSONObject)
 readJsonSimpleDemo(fileLocation);
            CreateRoleRequest request = CreateRoleRequest.builder()
                    .roleName(rolename)
                    .assumeRolePolicyDocument(jsonObject.toJSONString())
                    .description("Created using the AWS SDK for Java")
                    .build();

            CreateRoleResponse response = iam.createRole(request);
            System.out.println("The ARN of the role is " +
 response.role().arn());

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static Object readJsonSimpleDemo(String filename) throws Exception {
        FileReader reader = new FileReader(filename);
        JSONParser jsonParser = new JSONParser();
        return jsonParser.parse(reader);
    }
}
```

- For API details, see CreateRole in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the role.

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
  const command = new CreateRoleCommand({
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            Service: "lambda.amazonaws.com",
          },
          Action: "sts:AssumeRole",
        },
      ],
    }),
    RoleName: roleName,
  });

  return client.send(command);
};
```

- For API details, see CreateRole in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$assumeRolePolicyDocument = "{
                \"Version\": \"2012-10-17\",
                \"Statement\": [{
                    \"Effect\": \"Allow\",
                    \"Principal\": {\"AWS\": \"{$user['Arn']}\"},
                    \"Action\": \"sts:AssumeRole\"
                }]
            }";
$assumeRoleRole = $service->createRole("iam_demo_role_$uuid",
 $assumeRolePolicyDocument);
echo "Created role: {$assumeRoleRole['RoleName']}\n";

    /**
     * @param string $roleName
     * @param string $rolePolicyDocument
     * @return array
     * @throws AwsException
     */
    public function createRole(string $roleName, string $rolePolicyDocument)
    {
        $result = $this->customWaiter(function () use ($roleName,
 $rolePolicyDocument) {
            return $this->iamClient->createRole([
                'AssumeRolePolicyDocument' => $rolePolicyDocument,
                'RoleName' => $roleName,
            ]);
        });
        return $result['Role'];
    }
```

- For API details, see CreateRole in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new role named `MyNewRole` and attaches to it the policy found in the file `NewRoleTrustPolicy.json`. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file. The policy document displayed in the output is URL encoded. It is decoded in this example with the `UrlDecode` .NET method.

```
$results = New-IAMRole -AssumeRolePolicyDocument (Get-Content -raw
 NewRoleTrustPolicy.json) -RoleName MyNewRole
$results
```

Output:

```
Arn                       : arn:aws:iam::123456789012:role/MyNewRole
AssumeRolePolicyDocument : %7B%0D%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C
%0D%0A%20%20%22Statement%22
                           %3A%20%5B%0D%0A%20%20%20%20%7B%0D%0A
%20%20%20%20%20%20%22Sid%22%3A%20%22%22%2C
                           %0D%0A%20%20%20%20%20%20%22Effect%22%3A%20%22Allow
%22%2C%0D%0A%20%20%20%20%20%20
                           %22Principal%22%3A%20%7B%0D%0A
%20%20%20%20%20%20%20%20%22AWS%22%3A%20%22arn%3Aaws
                           %3Aiam%3A%3A123456789012%3ADavid%22%0D%0A
%20%20%20%20%20%20%7D%2C%0D%0A%20%20%20
                           %20%20%20%22Action%22%3A%20%22sts%3AAssumeRole%22%0D
%0A%20%20%20%20%7D%0D%0A%20
                           %20%5D%0D%0A%7D
CreateDate                : 4/15/2015 11:04:23 AM
Path                      : /
RoleId                    : V5PAJI2KPN4EAEXAMPLE1
RoleName                  : MyNewRole

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
```

```
[System.Web.HttpUtility]::UrlDecode($results.AssumeRolePolicyDocument)
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:David"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- For API details, see CreateRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new role named `MyNewRole` and attaches to it the policy found in the file `NewRoleTrustPolicy.json`. Note that you must use the `-Raw` switch parameter to successfully process the JSON policy file. The policy document displayed in the output is URL encoded. It is decoded in this example with the `UrlDecode` .NET method.

```
$results = New-IAMRole -AssumeRolePolicyDocument (Get-Content -raw
 NewRoleTrustPolicy.json) -RoleName MyNewRole
$results
```

Output:

```
Arn                       : arn:aws:iam::123456789012:role/MyNewRole
AssumeRolePolicyDocument : %7B%0D%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C
%0D%0A%20%20%22Statement%22
                           %3A%20%5B%0D%0A%20%20%20%20%7B%0D%0A
%20%20%20%20%20%20%22Sid%22%3A%20%22%22%2C
                           %0D%0A%20%20%20%20%20%20%22Effect%22%3A%20%22Allow
%22%2C%0D%0A%20%20%20%20%20%20
                           %22Principal%22%3A%20%7B%0D%0A
%20%20%20%20%20%20%20%20%22AWS%22%3A%20%22arn%3Aaws
                           %3Aiam%3A%3A123456789012%3ADavid%22%0D%0A
%20%20%20%20%20%20%7D%2C%0D%0A%20%20%20
```

```
                              %20%20%20%22Action%22%3A%20%22sts%3AAssumeRole%22%0D
%0A%20%20%20%20%7D%0D%0A%20
                              %20%5D%0D%0A%7D
CreateDate               : 4/15/2015 11:04:23 AM
Path                     : /
RoleId                   : V5PAJI2KPN4EAEXAMPLE1
RoleName                 : MyNewRole

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
[System.Web.HttpUtility]::UrlDecode($results.AssumeRolePolicyDocument)
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:David"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- For API details, see CreateRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_role(role_name, allowed_services):
    """
    Creates a role that lets a list of specified services assume the role.

    :param role_name: The name of the role.
    :param allowed_services: The services that can assume the role.
```

```
    :return: The newly created role.
    """
    trust_policy = {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {"Service": service},
                "Action": "sts:AssumeRole",
            }
            for service in allowed_services
        ],
    }

    try:
        role = iam.create_role(
            RoleName=role_name, AssumeRolePolicyDocument=json.dumps(trust_policy)
        )
        logger.info("Created role %s.", role.name)
    except ClientError:
        logger.exception("Couldn't create role %s.", role_name)
        raise
    else:
        return role
```

- For API details, see CreateRole in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Creates a role and attaches policies to it.
#
```

```
  # @param role_name [String] The name of the role.
  # @param assume_role_policy_document [Hash] The trust relationship policy
document.
  # @param policy_arns [Array<String>] The ARNs of the policies to attach.
  # @return [String, nil] The ARN of the new role if successful, or nil if an
error occurred.
  def create_role(role_name, assume_role_policy_document, policy_arns)
    response = @iam_client.create_role(
      role_name: role_name,
      assume_role_policy_document: assume_role_policy_document.to_json
    )
    role_arn = response.role.arn

    policy_arns.each do |policy_arn|
      @iam_client.attach_role_policy(
        role_name: role_name,
        policy_arn: policy_arn
      )
    end

    role_arn
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating role: #{e.message}")
    nil
  end
```

- For API details, see CreateRole in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn create_role(
    client: &iamClient,
    role_name: &str,
```

```
    role_policy_document: &str,
) -> Result<Role, iamError> {
    let response: CreateRoleOutput = loop {
        if let Ok(response) = client
            .create_role()
            .role_name(role_name)
            .assume_role_policy_document(role_policy_document)
            .send()
            .await
        {
            break response;
        }
    };

    Ok(response.role.unwrap())
}
```

- For API details, see CreateRole in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->createrole(
          iv_rolename = iv_role_name
          iv_assumerolepolicydocument = iv_assume_role_policy_document ).
        MESSAGE 'Role created successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamentityalrdyexex.
        MESSAGE 'Role already exists.' TYPE 'E'.
      CATCH /aws1/cx_iammalformedplydocex.
        MESSAGE 'Assume role policy document is malformed.' TYPE 'E'.
      CATCH /aws1/cx_iamlimitexceededex.
        MESSAGE 'Role limit exceeded.' TYPE 'E'.
    ENDTRY.
```

- For API details, see CreateRole in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func createRole(name: String, policyDocument: String) async throws ->
 String {
        let input = CreateRoleInput(
            assumeRolePolicyDocument: policyDocument,
            roleName: name
        )
        do {
            let output = try await client.createRole(input: input)
            guard let role = output.role else {
                throw ServiceHandlerError.noSuchRole
            }
            guard let id = role.roleId else {
                throw ServiceHandlerError.noSuchRole
            }
            return id
        } catch {
            print("ERROR: createRole:", dump(error))
            throw error
        }
    }
```

- For API details, see CreateRole in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateSAMLProvider` with an AWS SDK or CLI

The following code examples show how to use `CreateSAMLProvider`.

CLI

AWS CLI

To create a SAML provider

The following `create-saml-provider` example creates a new SAML provider in IAM named `MySAMLProvider`. It is described by the SAML metadata document found in the file `SAMLMetaData.xml`.

```
aws iam create-saml-provider \
    --saml-metadata-document file://SAMLMetaData.xml \
    --name MySAMLProvider
```

Contents of `SAMLMetaData.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
 entityID="https://idp.example.com/saml/metadata">
    <md:IDPSSODescriptor
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <md:KeyDescriptor use="signing">
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:X509Data>
                    <ds:X509Certificate>MIIC8DCCAdigAwIBAgIQEXAMPLE...</
ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </md:KeyDescriptor>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</
md:NameIDFormat>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</
md:NameIDFormat>
```

```
        <md:SingleSignOnService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect" Location="https://
idp.example.com/saml/sso"/>
        <md:SingleSignOnService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://
idp.example.com/saml/sso"/>
    </md:IDPSSODescriptor>
</md:EntityDescriptor>
```

Output:

```
{
    "SAMLProviderArn": "arn:aws:iam::123456789012:saml-provider/MySAMLProvider"
}
```

For more information, see Creating IAM SAML identity providers in the *AWS IAM User Guide*.

- For API details, see CreateSAMLProvider in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import * as path from "node:path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
 * For more information on generating this document,
 see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
 */
```

```
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For API details, see CreateSAMLProvider in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new SAML provider entity in IAM. It is named `MySAMLProvider` and is described by the SAML metadata document found in the file `SAMLMetaData.xml`, which was separately downloaded from the SAML service provider's web site.

```
New-IAMSAMLProvider -Name MySAMLProvider -SAMLMetadataDocument (Get-Content -Raw
 SAMLMetaData.xml)
```

Output:

```
arn:aws:iam::123456789012:saml-provider/MySAMLProvider
```

- For API details, see CreateSAMLProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new SAML provider entity in IAM. It is named `MySAMLProvider` and is described by the SAML metadata document found in the file `SAMLMetaData.xml`, which was separately downloaded from the SAML service provider's web site.

```
New-IAMSAMLProvider -Name MySAMLProvider -SAMLMetadataDocument (Get-Content -Raw
 SAMLMetaData.xml)
```

Output:

```
arn:aws:iam::123456789012:saml-provider/MySAMLProvider
```

- For API details, see CreateSAMLProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateServiceLinkedRole` with an AWS SDK or CLI

The following code examples show how to use `CreateServiceLinkedRole`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create an IAM service-linked role.
```

```
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName,
 string description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To create a service-linked role

The following `create-service-linked-role` example creates a service-linked role for the specified AWS service and attaches the specified description.

```
aws iam create-service-linked-role \
    --aws-service-name lex.amazonaws.com \
    --description "My service-linked role to support Lex"
```

Output:

```
{
    "Role": {
        "Path": "/aws-service-role/lex.amazonaws.com/",
        "RoleName": "AWSServiceRoleForLexBots",
        "RoleId": "AROA1234567890EXAMPLE",
```

```
        "Arn": "arn:aws:iam::1234567890:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
        "CreateDate": "2019-04-17T20:34:14+00:00",
        "AssumeRolePolicyDocument": {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Action": [
                        "sts:AssumeRole"
                    ],
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "lex.amazonaws.com"
                        ]
                    }
                }
            ]
        }
    }
}
```

For more information, see Using service-linked roles in the *AWS IAM User Guide*.

- For API details, see CreateServiceLinkedRole in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// CreateServiceLinkedRole creates a service-linked role that is owned by the
 specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(ctx context.Context,
 serviceName string, description string) (
 *types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.CreateServiceLinkedRole(ctx,
 &iam.CreateServiceLinkedRoleInput{
  AWSServiceName: aws.String(serviceName),
  Description:    aws.String(description),
 })
 if err != nil {
  log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
 serviceName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a service-linked role.

```
import {
  CreateServiceLinkedRoleCommand,
  GetRoleCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-
services-that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/
general/latest/gr/aws-service-information.html.
    AWSServiceName: serviceName,
  });
  try {
    const response = await client.send(command);
    console.log(response);
    return response;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInputException" &&
      caught.message.includes(
```

```
        "Service role name AWSServiceRoleForElasticBeanstalk has been taken in
 this account",
      )
    ) {
      console.warn(caught.message);
      return client.send(
        new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
      );
    }
    throw caught;
  }
};
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function createServiceLinkedRole($awsServiceName, $customSuffix = "",
 $description = "")
    {
        $createServiceLinkedRoleArguments = ['AWSServiceName' =>
 $awsServiceName];
        if ($customSuffix) {
            $createServiceLinkedRoleArguments['CustomSuffix'] = $customSuffix;
        }
        if ($description) {
            $createServiceLinkedRoleArguments['Description'] = $description;
        }
        return $this->iamClient-
>createServiceLinkedRole($createServiceLinkedRoleArguments);
```

```
    }
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a servicelinked role for autoscaling service.

```
New-IAMServiceLinkedRole -AWSServiceName autoscaling.amazonaws.com -CustomSuffix
 RoleNameEndsWithThis -Description "My service-linked role to support
 autoscaling"
```

- For API details, see CreateServiceLinkedRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a servicelinked role for autoscaling service.

```
New-IAMServiceLinkedRole -AWSServiceName autoscaling.amazonaws.com -CustomSuffix
 RoleNameEndsWithThis -Description "My service-linked role to support
 autoscaling"
```

- For API details, see CreateServiceLinkedRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_service_linked_role(service_name, description):
```

```
    """
    Creates a service-linked role.

    :param service_name: The name of the service that owns the role.
    :param description: A description to give the role.
    :return: The newly created role.
    """
    try:
        response = iam.meta.client.create_service_linked_role(
            AWSServiceName=service_name, Description=description
        )
        role = iam.Role(response["Role"]["RoleName"])
        logger.info("Created service-linked role %s.", role.name)
    except ClientError:
        logger.exception("Couldn't create service-linked role for %s.",
service_name)
        raise
    else:
        return role
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Creates a service-linked role
#
# @param service_name [String] The service name to create the role for.
# @param description [String] The description of the service-linked role.
# @param suffix [String] Suffix for customizing role name.
# @return [String] The name of the created role
def create_service_linked_role(service_name, description, suffix)
```

```
    response = @iam_client.create_service_linked_role(
      aws_service_name: service_name, description: description, custom_suffix:
 suffix
    )
    role_name = response.role.role_name
    @logger.info("Created service-linked role #{role_name}.")
    role_name
  rescue Aws::Errors::ServiceError => e
    @logger.error("Couldn't create service-linked role for #{service_name}.
 Here's why:")
    @logger.error("\t#{e.code}: #{e.message}")
    raise
  end
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn create_service_linked_role(
    client: &iamClient,
    aws_service_name: String,
    custom_suffix: Option<String>,
    description: Option<String>,
) -> Result<CreateServiceLinkedRoleOutput,
 SdkError<CreateServiceLinkedRoleError>> {
    let response = client
        .create_service_linked_role()
        .aws_service_name(aws_service_name)
        .set_custom_suffix(custom_suffix)
        .set_description(description)
        .send()
        .await?;
```

```
    Ok(response)
}
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listpolicyversions(
          iv_policyarn = iv_policy_arn ).
        MESSAGE 'Retrieved policy versions list.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Policy does not exist.' TYPE 'E'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when listing policy versions.' TYPE 'E'.
    ENDTRY.
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func createServiceLinkedRole(service: String, suffix: String? = nil,
 description: String?)
                    async throws -> IAMClientTypes.Role {
        let input = CreateServiceLinkedRoleInput(
            awsServiceName: service,
            customSuffix: suffix,
            description: description
        )
        do {
            let output = try await client.createServiceLinkedRole(input: input)
            guard let role = output.role else {
                throw ServiceHandlerError.noSuchRole
            }
            return role
        } catch {
            print("ERROR: createServiceLinkedRole:", dump(error))
            throw error
        }
    }
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateUser` with an AWS SDK or CLI

The following code examples show how to use `CreateUser`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Create read-only and read-write users

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }
```

- For API details, see CreateUser in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
```

```
# the global variable $VERBOSE is set to true.
########################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

########################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

########################################################################################
# function iam_create_user
#
# This function creates the specified IAM user, unless
# it already exists.
#
# Parameters:
#       -u user_name  -- The name of the user to create.
#
# Returns:
#       The ARN of the user.
#     And:
#       0 - If successful.
#       1 - If it fails.
########################################################################################
function iam_create_user() {
  local user_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_create_user"
    echo "Creates an AWS Identity and Access Management (IAM) user. You must
 supply a username:"
    echo "  -u user_name    The name of the user. It must be unique within the
 account."
    echo ""
```

```
}

# Retrieve the calling parameters.
while getopts "u:h" option; do
  case "${option}" in
    u) user_name="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
  errecho "ERROR: You must provide a username with the -u parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    User name:   $user_name"
iecho ""

# If the user already exists, we don't want to try to create it.
if (iam_user_exists "$user_name"); then
  errecho "ERROR: A user with that name already exists in the account."
  return 1
fi

response=$(aws iam create-user --user-name "$user_name" \
  --output text \
  --query 'User.Arn')

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports create-user operation failed.$response"
```

```
    return 1
  fi

  echo "$response"

  return 0
}
```

- For API details, see CreateUser in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::CreateUserRequest create_request;
    create_request.SetUserName(userName);

    auto create_outcome = iam.CreateUser(create_request);
    if (!create_outcome.IsSuccess()) {
        std::cerr << "Error creating IAM user " << userName << ":" <<
                  create_outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully created IAM user " << userName << std::endl;
    }

    return create_outcome.IsSuccess();
```

- For API details, see CreateUser in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To create an IAM user

The following `create-user` example creates an IAM user named Bob in the current account.

```
aws iam create-user \
    --user-name Bob
```

Output:

```
{
    "User": {
        "UserName": "Bob",
        "Path": "/",
        "CreateDate": "2023-06-08T03:20:41.270Z",
        "UserId": "AIDAIOSFODNN7EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:user/Bob"
    }
}
```

For more information, see Creating an IAM user in your AWS account in the *AWS IAM User Guide*.

Example 2: To create an IAM user at a specified path

The following `create-user` example creates an IAM user named Bob at the specified path.

```
aws iam create-user \
    --user-name Bob \
    --path /division_abc/subdivision_xyz/
```

Output:

```
{
    "User": {
        "Path": "/division_abc/subdivision_xyz/",
        "UserName": "Bob",
        "UserId": "AIDAIOSFODNN7EXAMPLE",
```

```
        "Arn": "arn:aws:iam::12345678012:user/division_abc/subdivision_xyz/Bob",
        "CreateDate": "2023-05-24T18:20:17+00:00"
    }
}
```

For more information, see IAM identifiers in the *AWS IAM User Guide*.

Example 3: To create an IAM user with tags

The following `create-user` example creates an IAM user named Bob with tags. This example uses the `--tags` parameter flag with the following JSON-formatted tags: `'{"Key": "Department", "Value": "Accounting"}' '{"Key": "Location", "Value": "Seattle"}'`. Alternatively, the `--tags` flag can be used with tags in the shorthand format: `'Key=Department,Value=Accounting Key=Location,Value=Seattle'`.

```
aws iam create-user \
    --user-name Bob \
    --tags '{"Key": "Department", "Value": "Accounting"}' '{"Key": "Location",
 "Value": "Seattle"}'
```

Output:

```
{
    "User": {
        "Path": "/",
        "UserName": "Bob",
        "UserId": "AIDAIOSFODNN7EXAMPLE",
        "Arn": "arn:aws:iam::12345678012:user/Bob",
        "CreateDate": "2023-05-25T17:14:21+00:00",
        "Tags": [
            {
                "Key": "Department",
                "Value": "Accounting"
            },
            {
                "Key": "Location",
                "Value": "Seattle"
            }
        ]
    }
}
```

For more information, see Tagging IAM users in the *AWS IAM User Guide*.

Example 4: To create an IAM user with a set permissions boundary

The following `create-user` example creates an IAM user named Bob with the permissions boundary of AmazonS3FullAccess.

```
aws iam create-user \
    --user-name Bob \
    --permissions-boundary arn:aws:iam::aws:policy/AmazonS3FullAccess
```

Output:

```
{
    "User": {
        "Path": "/",
        "UserName": "Bob",
        "UserId": "AIDAIOSFODNN7EXAMPLE",
        "Arn": "arn:aws:iam::12345678012:user/Bob",
        "CreateDate": "2023-05-24T17:50:53+00:00",
        "PermissionsBoundary": {
        "PermissionsBoundaryType": "Policy",
        "PermissionsBoundaryArn": "arn:aws:iam::aws:policy/AmazonS3FullAccess"
        }
    }
}
```

For more information, see Permissions boundaries for IAM entities in the *AWS IAM User Guide*.

- For API details, see CreateUser in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.CreateUser(ctx, &iam.CreateUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
 } else {
  user = result.User
 }
 return user, err
}
```

- For API details, see CreateUser in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.iam.model.CreateUserRequest;
import software.amazon.awssdk.services.iam.model.CreateUserResponse;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.waiters.IamWaiter;
import software.amazon.awssdk.services.iam.model.GetUserRequest;
import software.amazon.awssdk.services.iam.model.GetUserResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateUser {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <username>\s

                Where:
                    username - The name of the user to create.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String username = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        String result = createIAMUser(iam, username);
        System.out.println("Successfully created user: " + result);
        iam.close();
    }

    public static String createIAMUser(IamClient iam, String username) {
        try {
            // Create an IamWaiter object.
            IamWaiter iamWaiter = iam.waiter();

            CreateUserRequest request = CreateUserRequest.builder()
                    .userName(username)
                    .build();

            CreateUserResponse response = iam.createUser(request);

            // Wait until the user is created.
            GetUserRequest userRequest = GetUserRequest.builder()
                    .userName(response.user().userName())
                    .build();

            WaiterResponse<GetUserResponse> waitUntilUserExists =
 iamWaiter.waitUntilUserExists(userRequest);

 waitUntilUserExists.matched().response().ifPresent(System.out::println);
            return response.user().userName();

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see CreateUser in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the user.

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateUser in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateUser in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createIAMUser(usernameVal: String?): String? {
    val request =
        CreateUserRequest {
            userName = usernameVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.createUser(request)
        return response.user?.userName
    }
}
```

- For API details, see CreateUser in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

$user = $service->createUser("iam_demo_user_$uuid");
echo "Created user with the arn: {$user['Arn']}\n";


    /**
     * @param string $name
     * @return array
     * @throws AwsException
     */
    public function createUser(string $name): array
    {
        $result = $this->iamClient->createUser([
            'UserName' => $name,
```

```
        ]);

        return $result['User'];
    }
```

- For API details, see CreateUser in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates an IAM user named Bob. If Bob needs to sign in to the AWS console, then you must separately run the command `New-IAMLoginProfile` to create a sign-in profile with a password. If Bob needs to run AWS PowerShell or cross-platform CLI commands or make AWS API calls, then you must separately run the `New-IAMAccessKey` command to create access keys.

```
New-IAMUser -UserName Bob
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Bob
CreateDate       : 4/22/2015 12:02:11 PM
PasswordLastUsed : 1/1/0001 12:00:00 AM
Path             : /
UserId           : AIDAJWGEFDMEMEXAMPLE1
UserName         : Bob
```

- For API details, see CreateUser in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates an IAM user named Bob. If Bob needs to sign in to the AWS console, then you must separately run the command `New-IAMLoginProfile` to create a sign-in profile with a password. If Bob needs to run AWS PowerShell or cross-platform CLI commands or make AWS API calls, then you must separately run the `New-IAMAccessKey` command to create access keys.

```
New-IAMUser -UserName Bob
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Bob
CreateDate       : 4/22/2015 12:02:11 PM
PasswordLastUsed : 1/1/0001 12:00:00 AM
Path             : /
UserId           : AIDAJWGEFDMEMEXAMPLE1
UserName         : Bob
```

- For API details, see CreateUser in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_user(user_name):
    """
    Creates a user. By default, a user has no permissions or access keys.

    :param user_name: The name of the user.
    :return: The newly created user.
    """
    try:
        user = iam.create_user(UserName=user_name)
        logger.info("Created user %s.", user.name)
    except ClientError:
        logger.exception("Couldn't create user %s.", user_name)
        raise
    else:
        return user
```

- For API details, see CreateUser in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
 # Creates a user and their login profile
 #
 # @param user_name [String] The name of the user
 # @param initial_password [String] The initial password for the user
 # @return [String, nil] The ID of the user if created, or nil if an error
occurred
 def create_user(user_name, initial_password)
   response = @iam_client.create_user(user_name: user_name)
   @iam_client.wait_until(:user_exists, user_name: user_name)
   @iam_client.create_login_profile(
     user_name: user_name,
     password: initial_password,
     password_reset_required: true
   )
   @logger.info("User '#{user_name}' created successfully.")
   response.user.user_id
 rescue Aws::IAM::Errors::EntityAlreadyExists
   @logger.error("Error creating user '#{user_name}': user already exists.")
   nil
 rescue Aws::IAM::Errors::ServiceError => e
   @logger.error("Error creating user '#{user_name}': #{e.message}")
   nil
 end
```

- For API details, see CreateUser in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn create_user(client: &iamClient, user_name: &str) -> Result<User,
 iamError> {
    let response = client.create_user().user_name(user_name).send().await?;

    Ok(response.user.unwrap())
}
```

- For API details, see CreateUser in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->createuser(
          iv_username = iv_user_name ).
        MESSAGE 'User created successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamentityalrdyexex.
        MESSAGE 'User already exists.' TYPE 'E'.
      CATCH /aws1/cx_iamlimitexceededex.
        MESSAGE 'Limit exceeded for IAM users.' TYPE 'E'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Entity does not exist.' TYPE 'E'.
```

```
    ENDTRY.
```

- For API details, see CreateUser in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func createUser(name: String) async throws -> String {
        let input = CreateUserInput(
            userName: name
        )
        do {
            let output = try await client.createUser(input: input)
            guard let user = output.user else {
                throw ServiceHandlerError.noSuchUser
            }
            guard let id = user.userId else {
                throw ServiceHandlerError.noSuchUser
            }
            return id
        } catch {
            print("ERROR: createUser:", dump(error))
            throw error
        }
    }
```

- For API details, see CreateUser in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateVirtualMfaDevice` with a CLI

The following code examples show how to use `CreateVirtualMfaDevice`.

CLI

AWS CLI

To create a virtual MFA device

The following `create-virtual-mfa-device` example creates a new virtual MFA device called `BobsMFADevice`. It creates a file that contains bootstrap information called `QRCode.png` and places it in the `C:/` directory. The bootstrap method used in this example is `QRCodePNG`.

```
aws iam create-virtual-mfa-device \
    --virtual-mfa-device-name BobsMFADevice \
    --outfile C:/QRCode.png \
    --bootstrap-method QRCodePNG
```

Output:

```
{
    "VirtualMFADevice": {
        "SerialNumber": "arn:aws:iam::123456789012:mfa/BobsMFADevice"
    }
}
```

For more information, see Using multi-factor authentication (MFA) in AWS in the *AWS IAM User Guide*.

- For API details, see CreateVirtualMfaDevice in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new virtual MFA device. Lines 2 and 3 extract the `Base32StringSeed` value that the virtual MFA software program needs to create an account (as an alternative to the QR code). After you configure the program with the value, get two sequential authentication codes from the program. Finally, use the last command to link the virtual MFA device to the IAM user Bob and synchronize the account with the two authentication codes.

```
$Device = New-IAMVirtualMFADevice -VirtualMFADeviceName BobsMFADevice
$SR = New-Object System.IO.StreamReader($Device.Base32StringSeed)
$base32stringseed = $SR.ReadToEnd()
$base32stringseed
CZWZMCQNW4DEXAMPLE3VOUGXJFZYSUW7EXAMPLECR4NJFD65GX2SLUDW2EXAMPLE
```

Output:

```
-- Pause here to enter base-32 string seed code into virtual MFA program to
 register account. --

Enable-IAMMFADevice -SerialNumber $Device.SerialNumber -UserName Bob -
AuthenticationCode1 123456 -AuthenticationCode2 789012
```

Example 2: This example creates a new virtual MFA device. Lines 2 and 3 extract the `QRCodePNG` value and write it to a file. This image can be scanned by the virtual MFA software program to create an account (as an alternative to manually entering the Base32StringSeed value). After you create the account in your virtual MFA program, get two sequential authentication codes and enter them in the last commands to link the virtual MFA device to the IAM user Bob and synchronize the account.

```
$Device = New-IAMVirtualMFADevice -VirtualMFADeviceName BobsMFADevice
$BR = New-Object System.IO.BinaryReader($Device.QRCodePNG)
$BR.ReadBytes($BR.BaseStream.Length) | Set-Content -Encoding Byte -Path
 QRCode.png
```

Output:

```
 -- Pause here to scan PNG with virtual MFA program to register account. --
```

```
Enable-IAMMFADevice -SerialNumber $Device.SerialNumber -UserName Bob -
AuthenticationCode1 123456 -AuthenticationCode2 789012
```

- For API details, see CreateVirtualMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new virtual MFA device. Lines 2 and 3 extract the `Base32StringSeed` value that the virtual MFA software program needs to create an account (as an alternative to the QR code). After you configure the program with the value, get two sequential authentication codes from the program. Finally, use the last command to link the virtual MFA device to the IAM user Bob and synchronize the account with the two authentication codes.

```
$Device = New-IAMVirtualMFADevice -VirtualMFADeviceName BobsMFADevice
$SR = New-Object System.IO.StreamReader($Device.Base32StringSeed)
$base32stringseed = $SR.ReadToEnd()
$base32stringseed
CZWZMCQNW4DEXAMPLE3VOUGXJFZYSUW7EXAMPLECR4NJFD65GX2SLUDW2EXAMPLE
```

Output:

```
-- Pause here to enter base-32 string seed code into virtual MFA program to
 register account. --

Enable-IAMMFADevice -SerialNumber $Device.SerialNumber -UserName Bob -
AuthenticationCode1 123456 -AuthenticationCode2 789012
```

Example 2: This example creates a new virtual MFA device. Lines 2 and 3 extract the `QRCodePNG` value and write it to a file. This image can be scanned by the virtual MFA software program to create an account (as an alternative to manually entering the Base32StringSeed value). After you create the account in your virtual MFA program, get two sequential authentication codes and enter them in the last commands to link the virtual MFA device to the IAM user Bob and synchronize the account.

```
$Device = New-IAMVirtualMFADevice -VirtualMFADeviceName BobsMFADevice
$BR = New-Object System.IO.BinaryReader($Device.QRCodePNG)
```

```
$BR.ReadBytes($BR.BaseStream.Length) | Set-Content -Encoding Byte -Path
 QRCode.png
```

Output:

```
 -- Pause here to scan PNG with virtual MFA program to register account. --

Enable-IAMMFADevice -SerialNumber $Device.SerialNumber -UserName Bob -
AuthenticationCode1 123456 -AuthenticationCode2 789012
```

- For API details, see CreateVirtualMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeactivateMfaDevice` with a CLI

The following code examples show how to use `DeactivateMfaDevice`.

CLI

AWS CLI

To deactivate an MFA device

The following `deactivate-mfa-device` example deactivates the virtual MFA device with the ARN `arn:aws:iam::123456789012:mfa/BobsMFADevice` that is associated with the user Bob.

```
aws iam deactivate-mfa-device \
    --user-name Bob \
    --serial-number arn:aws:iam::123456789012:mfa/BobsMFADevice
```

This command produces no output.

For more information, see Using multi-factor authentication (MFA) in AWS in the *AWS IAM User Guide*.

- For API details, see DeactivateMfaDevice in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command disables the hardware MFA device associated with the user Bob that has the serial number 123456789012.

```
Disable-IAMMFADevice -UserName "Bob" -SerialNumber "123456789012"
```

Example 2: This command disables the virtual MFA device associated with the user David that has the ARN arn:aws:iam::210987654321:mfa/David. Note that virtual MFA device is not deleted from the account. The virtual device is still present and appears in the output of the Get-IAMVirtualMFADevice command. Before you can create a new virtual MFA device for the same user, you must delete the old one by using the Remove-IAMVirtualMFADevice command.

```
Disable-IAMMFADevice -UserName "David" -SerialNumber
 "arn:aws:iam::210987654321:mfa/David"
```

- For API details, see DeactivateMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command disables the hardware MFA device associated with the user Bob that has the serial number 123456789012.

```
Disable-IAMMFADevice -UserName "Bob" -SerialNumber "123456789012"
```

Example 2: This command disables the virtual MFA device associated with the user David that has the ARN arn:aws:iam::210987654321:mfa/David. Note that virtual MFA device is not deleted from the account. The virtual device is still present and appears in the output of the Get-IAMVirtualMFADevice command. Before you can create a new virtual MFA device for the same user, you must delete the old one by using the Remove-IAMVirtualMFADevice command.

```
Disable-IAMMFADevice -UserName "David" -SerialNumber
 "arn:aws:iam::210987654321:mfa/David"
```

- For API details, see DeactivateMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAccessKey` with an AWS SDK or CLI

The following code examples show how to use `DeleteAccessKey`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Create read-only and read-write users
- Manage access keys

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
```

```
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DeleteAccessKey in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function iam_delete_access_key
#
# This function deletes an IAM access key for the specified IAM user.
#
# Parameters:
#       -u user_name  -- The name of the user.
#       -k access_key -- The access key to delete.
#
# Returns:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function iam_delete_access_key() {
  local user_name access_key response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_access_key"
    echo "Deletes an AWS Identity and Access Management (IAM) access key for the
 specified IAM user"
    echo "  -u user_name    The name of the user."
    echo "  -k access_key   The access key to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "u:k:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      k) access_key="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  if [[ -z "$access_key" ]]; then
    errecho "ERROR: You must provide an access key with the -k parameter."
    usage
    return 1
```

```
  fi

  iecho "Parameters:\n"
  iecho "    Username:   $user_name"
  iecho "    Access key:   $access_key"
  iecho ""

  response=$(aws iam delete-access-key \
    --user-name "$user_name" \
    --access-key-id "$access_key")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-access-key operation failed.\n$response"
    return 1
  fi

  iecho "delete-access-key response:$response"
  iecho

  return 0
}
```

- For API details, see DeleteAccessKey in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::deleteAccessKey(const Aws::String &userName,
                                  const Aws::String &accessKeyID,
                                  const Aws::Client::ClientConfiguration
 &clientConfig) {
```

```
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::DeleteAccessKeyRequest request;
    request.SetUserName(userName);
    request.SetAccessKeyId(accessKeyID);

    auto outcome = iam.DeleteAccessKey(request);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error deleting access key " << accessKeyID << " from user "
                  << userName << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }
    else {
        std::cout << "Successfully deleted access key " << accessKeyID
                  << " for IAM user " << userName << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteAccessKey in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an access key for an IAM user

The following `delete-access-key` example deletes the specified access key (access key ID and secret access key) for the IAM user named Bob.

```
aws iam delete-access-key \
    --access-key-id AKIDPMS9RO4H3FEXAMPLE \
    --user-name Bob
```

This command produces no output.

To list the access keys defined for an IAM user, use the `list-access-keys` command.

For more information, see Managing access keys for IAM users in the *AWS IAM User Guide*.

- For API details, see DeleteAccessKey in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(ctx context.Context, userName string,
 keyId string) error {
 _, err := wrapper.IamClient.DeleteAccessKey(ctx, &iam.DeleteAccessKeyInput{
  AccessKeyId: aws.String(keyId),
  UserName:    aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
```

```
 }
 return err
}
```

- For API details, see DeleteAccessKey in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.DeleteAccessKeyRequest;
import software.amazon.awssdk.services.iam.model.IamException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteAccessKey {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <username> <accessKey>\s

                Where:
                    username - The name of the user.\s
```

```
                    accessKey - The access key ID for the secret access key you
 want to delete.\s
                """";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String username = args[0];
        String accessKey = args[1];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();
        deleteKey(iam, username, accessKey);
        iam.close();
    }

    public static void deleteKey(IamClient iam, String username, String
 accessKey) {
        try {
            DeleteAccessKeyRequest request = DeleteAccessKeyRequest.builder()
                    .accessKeyId(accessKey)
                    .userName(username)
                    .build();

            iam.deleteAccessKey(request);
            System.out.println("Successfully deleted access key " + accessKey +
                    " from user " + username);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteAccessKey in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete the access key.

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
    UserName: userName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteAccessKey in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteAccessKey in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteKey(
    userNameVal: String,
    accessKey: String,
) {
    val request =
```

```
        DeleteAccessKeyRequest {
            accessKeyId = accessKey
            userName = userNameVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deleteAccessKey(request)
        println("Successfully deleted access key $accessKey from $userNameVal")
    }
}
```

- For API details, see DeleteAccessKey in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the AWS access key pair with the key ID AKIAIOSFODNN7EXAMPLE from the user named Bob.

```
Remove-IAMAccessKey -AccessKeyId AKIAIOSFODNN7EXAMPLE -UserName Bob -Force
```

- For API details, see DeleteAccessKey in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the AWS access key pair with the key ID AKIAIOSFODNN7EXAMPLE from the user named Bob.

```
Remove-IAMAccessKey -AccessKeyId AKIAIOSFODNN7EXAMPLE -UserName Bob -Force
```

- For API details, see DeleteAccessKey in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def delete_key(user_name, key_id):
    """
    Deletes a user's access key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to delete.
    """

    try:
        key = iam.AccessKey(user_name, key_id)
        key.delete()
        logger.info("Deleted access key %s for %s.", key.id, key.user_name)
    except ClientError:
        logger.exception("Couldn't delete key %s for %s", key_id, user_name)
        raise
```

- For API details, see DeleteAccessKey in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, deactivates, and deletes access keys.

```
# Manages access keys for IAM users
class AccessKeyManager
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'AccessKeyManager'
  end

  # Lists access keys for a user
  #
  # @param user_name [String] The name of the user.
  def list_access_keys(user_name)
    response = @iam_client.list_access_keys(user_name: user_name)
    if response.access_key_metadata.empty?
      @logger.info("No access keys found for user '#{user_name}'.")
    else
      response.access_key_metadata.map(&:access_key_id)
    end
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Error listing access keys: cannot find user '#{user_name}'.")
    []
  rescue StandardError => e
    @logger.error("Error listing access keys: #{e.message}")
    []
  end

  # Creates an access key for a user
  #
  # @param user_name [String] The name of the user.
  # @return [Boolean]
  def create_access_key(user_name)
    response = @iam_client.create_access_key(user_name: user_name)
    access_key = response.access_key
    @logger.info("Access key created for user '#{user_name}':
 #{access_key.access_key_id}")
    access_key
  rescue Aws::IAM::Errors::LimitExceeded
    @logger.error('Error creating access key: limit exceeded. Cannot create
 more.')
    nil
  rescue StandardError => e
    @logger.error("Error creating access key: #{e.message}")
```

```
    nil
  end

  # Deactivates an access key
  #
  # @param user_name [String] The name of the user.
  # @param access_key_id [String] The ID for the access key.
  # @return [Boolean]
  def deactivate_access_key(user_name, access_key_id)
    @iam_client.update_access_key(
      user_name: user_name,
      access_key_id: access_key_id,
      status: 'Inactive'
    )
    true
  rescue StandardError => e
    @logger.error("Error deactivating access key: #{e.message}")
    false
  end

  # Deletes an access key
  #
  # @param user_name [String] The name of the user.
  # @param access_key_id [String] The ID for the access key.
  # @return [Boolean]
  def delete_access_key(user_name, access_key_id)
    @iam_client.delete_access_key(
      user_name: user_name,
      access_key_id: access_key_id
    )
    true
  rescue StandardError => e
    @logger.error("Error deleting access key: #{e.message}")
    false
  end
end
```

- For API details, see DeleteAccessKey in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn delete_access_key(
    client: &iamClient,
    user: &User,
    key: &AccessKey,
) -> Result<(), iamError> {
    loop {
        match client
            .delete_access_key()
            .user_name(user.user_name())
            .access_key_id(key.access_key_id())
            .send()
            .await
        {
            Ok(_) => {
                break;
            }
            Err(e) => {
                println!("Can't delete the access key: {:?}", e);
                sleep(Duration::from_secs(2)).await;
            }
        }
    }
    Ok(())
}
```

- For API details, see DeleteAccessKey in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->deleteaccesskey(
          iv_accesskeyid = iv_access_key_id
          iv_username = iv_user_name ).
        MESSAGE 'Access key deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Access key or user does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeleteAccessKey in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func deleteAccessKey(user: IAMClientTypes.User? = nil,
                                key: IAMClientTypes.AccessKey) async throws
    {
        let userName: String?
```

```
        if user != nil {
            userName = user!.userName
        } else {
            userName = nil
        }

        let input = DeleteAccessKeyInput(
            accessKeyId: key.accessKeyId,
            userName: userName
        )
        do {
            _ = try await iamClient.deleteAccessKey(input: input)
        } catch {
            print("ERROR: deleteAccessKey:", dump(error))
            throw error
        }
    }
```

- For API details, see DeleteAccessKey in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAccountAlias` with an AWS SDK or CLI

The following code examples show how to use `DeleteAccountAlias`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage your account

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::deleteAccountAlias(const Aws::String &accountAlias,
                                     const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::DeleteAccountAliasRequest request;
    request.SetAccountAlias(accountAlias);

    const auto outcome = iam.DeleteAccountAlias(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error deleting account alias " << accountAlias << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully deleted account alias " << accountAlias <<
                  std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeleteAccountAlias in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an account alias

The following `delete-account-alias` example removes the alias `mycompany` for the current account.

```
aws iam delete-account-alias \
    --account-alias mycompany
```

This command produces no output.

For more information, see Your AWS account ID and its alias in the *AWS IAM User Guide*.

- For API details, see DeleteAccountAlias in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.DeleteAccountAliasRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteAccountAlias {
    public static void main(String[] args) {
        final String usage = """

                Usage:
```

```
                    <alias>\s

                Where:
                    alias - The account alias to delete.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alias = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        deleteIAMAccountAlias(iam, alias);
        iam.close();
    }

    public static void deleteIAMAccountAlias(IamClient iam, String alias) {
        try {
            DeleteAccountAliasRequest request =
 DeleteAccountAliasRequest.builder()
                    .accountAlias(alias)
                    .build();

            iam.deleteAccountAlias(request);
            System.out.println("Successfully deleted account alias " + alias);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        System.out.println("Done");
    }
}
```

- For API details, see DeleteAccountAlias in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete the account alias.

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteAccountAlias in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteAccountAlias in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteIAMAccountAlias(alias: String) {
    val request =
        DeleteAccountAliasRequest {
            accountAlias = alias
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deleteAccountAlias(request)
        println("Successfully deleted account alias $alias")
    }
}
```

- For API details, see DeleteAccountAlias in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example removes the account alias from your AWS account. The user sign in page with the alias at https://mycompanyaws.signin.aws.amazon.com/console no longer works. You must instead use the original URL with your AWS account ID number at https://<accountidnumber>.signin.aws.amazon.com/console.

```
Remove-IAMAccountAlias -AccountAlias mycompanyaws
```

- For API details, see DeleteAccountAlias in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example removes the account alias from your AWS account. The user sign in page with the alias at https://mycompanyaws.signin.aws.amazon.com/console no longer works. You must instead use the original URL with your AWS account ID number at https://<accountidnumber>.signin.aws.amazon.com/console.

```
Remove-IAMAccountAlias -AccountAlias mycompanyaws
```

- For API details, see DeleteAccountAlias in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def delete_alias(alias):
    """
    Removes the alias from the current account.

    :param alias: The alias to remove.
    """
    try:
```

```
            iam.meta.client.delete_account_alias(AccountAlias=alias)
            logger.info("Removed alias '%s' from your account.", alias)
        except ClientError:
            logger.exception("Couldn't remove alias '%s' from your account.", alias)
            raise
```

- For API details, see DeleteAccountAlias in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List, create, and delete account aliases.

```
class IAMAliasManager
  # Initializes the IAM client and logger
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client.
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
  end

  # Lists available AWS account aliases.
  def list_aliases
    response = @iam_client.list_account_aliases

    if response.account_aliases.count.positive?
      @logger.info('Account aliases are:')
      response.account_aliases.each { |account_alias| @logger.info("
 #{account_alias}") }
    else
      @logger.info('No account aliases found.')
```

```
    end
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing account aliases: #{e.message}")
  end

  # Creates an AWS account alias.
  #
  # @param account_alias [String] The name of the account alias to create.
  # @return [Boolean] true if the account alias was created; otherwise, false.
  def create_account_alias(account_alias)
    @iam_client.create_account_alias(account_alias: account_alias)
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating account alias: #{e.message}")
    false
  end

  # Deletes an AWS account alias.
  #
  # @param account_alias [String] The name of the account alias to delete.
  # @return [Boolean] true if the account alias was deleted; otherwise, false.
  def delete_account_alias(account_alias)
    @iam_client.delete_account_alias(account_alias: account_alias)
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting account alias: #{e.message}")
    false
  end
end
```

- For API details, see DeleteAccountAlias in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->deleteaccountalias(
          iv_accountalias = iv_account_alias ).
        MESSAGE 'Account alias deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Account alias does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeleteAccountAlias in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAccountPasswordPolicy` with a CLI

The following code examples show how to use `DeleteAccountPasswordPolicy`.

CLI

AWS CLI

To delete the current account password policy

The following `delete-account-password-policy` example removes the password policy for the current account.

```
aws iam delete-account-password-policy
```

This command produces no output.

For more information, see Setting an account password policy for IAM users in the *AWS IAM User Guide*.

- For API details, see DeleteAccountPasswordPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the password policy for the AWS account and resets all values to their original defaults. If a password policy does not currently exist, the following error message appears: The account policy with name PasswordPolicy cannot be found.

```
Remove-IAMAccountPasswordPolicy
```

- For API details, see DeleteAccountPasswordPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the password policy for the AWS account and resets all values to their original defaults. If a password policy does not currently exist, the following error message appears: The account policy with name PasswordPolicy cannot be found.

```
Remove-IAMAccountPasswordPolicy
```

- For API details, see DeleteAccountPasswordPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteGroup` with an AWS SDK or CLI

The following code examples show how to use `DeleteGroup`.

CLI

AWS CLI

To delete an IAM group

The following `delete-group` example deletes an IAM group named `MyTestGroup`.

```
aws iam delete-group \
    --group-name MyTestGroup
```

This command produces no output.

For more information, see Deleting an IAM user group in the *AWS IAM User Guide*.

- For API details, see DeleteGroup in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For API details, see DeleteGroup in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the IAM group named `MyTestGroup`. The first command removes any IAM users that are members of the group, and the second command deletes the IAM group. Both commands work without any prompts for confirmation.

```
(Get-IAMGroup -GroupName MyTestGroup).Users | Remove-IAMUserFromGroup -GroupName
 MyTestGroup -Force
Remove-IAMGroup -GroupName MyTestGroup -Force
```

- For API details, see DeleteGroup in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the IAM group named `MyTestGroup`. The first command removes any IAM users that are members of the group, and the second command deletes the IAM group. Both commands work without any prompts for confirmation.

```
(Get-IAMGroup -GroupName MyTestGroup).Users | Remove-IAMUserFromGroup -GroupName
 MyTestGroup -Force
Remove-IAMGroup -GroupName MyTestGroup -Force
```

- For API details, see DeleteGroup in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteGroupPolicy` with a CLI

The following code examples show how to use `DeleteGroupPolicy`.

CLI

AWS CLI

To delete a policy from an IAM group

The following `delete-group-policy` example deletes the policy named `ExamplePolicy` from the group named `Admins`.

```
aws iam delete-group-policy \
    --group-name Admins \
    --policy-name ExamplePolicy
```

This command produces no output.

To see the policies attached to a group, use the `list-group-policies` command.

For more information, see Managing IAM policies in the *AWS IAM User Guide*.

- For API details, see DeleteGroupPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example removes the inline policy named `TesterPolicy` from the IAM group `Testers`. The users in that group immediately lose the permissions defined in that policy.

```
Remove-IAMGroupPolicy -GroupName Testers -PolicyName TestPolicy
```

- For API details, see DeleteGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example removes the inline policy named `TesterPolicy` from the IAM group `Testers`. The users in that group immediately lose the permissions defined in that policy.

```
Remove-IAMGroupPolicy -GroupName Testers -PolicyName TestPolicy
```

- For API details, see DeleteGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteInstanceProfile` with an AWS SDK or CLI

The following code examples show how to use `DeleteInstanceProfile`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Build and manage a resilient service
- Getting started with managed streaming
- Run CPU stress tests on virtual machine instances using fault injection

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the
role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await
_amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
```

```
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }
```

- For API details, see DeleteInstanceProfile in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To delete an instance profile

The following `delete-instance-profile` example deletes the instance profile named `ExampleInstanceProfile`.

```
aws iam delete-instance-profile \
    --instance-profile-name ExampleInstanceProfile
```

This command produces no output.

For more information, see Using instance profiles in the *AWS IAM User Guide*.

- For API details, see DeleteInstanceProfile in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const client = new IAMClient({});
await client.send(
  new DeleteInstanceProfileCommand({
    InstanceProfileName: NAMES.instanceProfileName,
  }),
);
```

- For API details, see DeleteInstanceProfile in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the EC2 instance profile named `MyAppInstanceProfile`. The first command detaches any roles from the instance profile, and then the second command deletes the instance profile.

```
(Get-IAMInstanceProfile -InstanceProfileName MyAppInstanceProfile).Roles |
 Remove-IAMRoleFromInstanceProfile -InstanceProfileName MyAppInstanceProfile
Remove-IAMInstanceProfile -InstanceProfileName MyAppInstanceProfile
```

- For API details, see DeleteInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the EC2 instance profile named `MyAppInstanceProfile`. The first command detaches any roles from the instance profile, and then the second command deletes the instance profile.

```
(Get-IAMInstanceProfile -InstanceProfileName MyAppInstanceProfile).Roles |
 Remove-IAMRoleFromInstanceProfile -InstanceProfileName MyAppInstanceProfile
Remove-IAMInstanceProfile -InstanceProfileName MyAppInstanceProfile
```

- For API details, see DeleteInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example removes the role from the instance profile, detaches all policies attached to the role, and deletes all the resources.

```
class AutoScalingWrapper:
    """
    Encapsulates Amazon EC2 Auto Scaling and EC2 management actions.
    """

    def __init__(
        self,
        resource_prefix: str,
        inst_type: str,
        ami_param: str,
        autoscaling_client: boto3.client,
        ec2_client: boto3.client,
        ssm_client: boto3.client,
        iam_client: boto3.client,
    ):
```

```
        """
        Initializes the AutoScaler class with the necessary parameters.

        :param resource_prefix: The prefix for naming AWS resources that are
created by this class.
        :param inst_type: The type of EC2 instance to create, such as t3.micro.
        :param ami_param: The Systems Manager parameter used to look up the AMI
that is created.
        :param autoscaling_client: A Boto3 EC2 Auto Scaling client.
        :param ec2_client: A Boto3 EC2 client.
        :param ssm_client: A Boto3 Systems Manager client.
        :param iam_client: A Boto3 IAM client.
        """
        self.inst_type = inst_type
        self.ami_param = ami_param
        self.autoscaling_client = autoscaling_client
        self.ec2_client = ec2_client
        self.ssm_client = ssm_client
        self.iam_client = iam_client
        sts_client = boto3.client("sts")
        self.account_id = sts_client.get_caller_identity()["Account"]

        self.key_pair_name = f"{resource_prefix}-key-pair"
        self.launch_template_name = f"{resource_prefix}-template-"
        self.group_name = f"{resource_prefix}-group"

        # Happy path
        self.instance_policy_name = f"{resource_prefix}-pol"
        self.instance_role_name = f"{resource_prefix}-role"
        self.instance_profile_name = f"{resource_prefix}-prof"

        # Failure mode
        self.bad_creds_policy_name = f"{resource_prefix}-bc-pol"
        self.bad_creds_role_name = f"{resource_prefix}-bc-role"
        self.bad_creds_profile_name = f"{resource_prefix}-bc-prof"


    def delete_instance_profile(self, profile_name: str, role_name: str) -> None:
        """
        Detaches a role from an instance profile, detaches policies from the
role,
        and deletes all the resources.

        :param profile_name: The name of the profile to delete.
```

```
        :param role_name: The name of the role to delete.
        """
        try:
            self.iam_client.remove_role_from_instance_profile(
                InstanceProfileName=profile_name, RoleName=role_name
            )

self.iam_client.delete_instance_profile(InstanceProfileName=profile_name)
            log.info("Deleted instance profile %s.", profile_name)
            attached_policies = self.iam_client.list_attached_role_policies(
                RoleName=role_name
            )
            for pol in attached_policies["AttachedPolicies"]:
                self.iam_client.detach_role_policy(
                    RoleName=role_name, PolicyArn=pol["PolicyArn"]
                )
                if not pol["PolicyArn"].startswith("arn:aws:iam::aws"):
                    self.iam_client.delete_policy(PolicyArn=pol["PolicyArn"])
                log.info("Detached and deleted policy %s.", pol["PolicyName"])
            self.iam_client.delete_role(RoleName=role_name)
            log.info("Deleted role %s.", role_name)
        except ClientError as err:
            log.error(
                f"Couldn't delete instance profile {profile_name} or detach "
                f"policies and delete role {role_name}: {err}"
            )
            if err.response["Error"]["Code"] == "NoSuchEntity":
                log.info(
                    "Instance profile %s doesn't exist, nothing to do.",
profile_name
                )
```

- For API details, see DeleteInstanceProfile in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteLoginProfile` with a CLI

The following code examples show how to use `DeleteLoginProfile`.

CLI

AWS CLI

To delete a password for an IAM user

The following `delete-login-profile` example deletes the password for the IAM user named Bob.

```
aws iam delete-login-profile \
    --user-name Bob
```

This command produces no output.

For more information, see Managing passwords for IAM users in the *AWS IAM User Guide*.

- For API details, see DeleteLoginProfile in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the login profile from the IAM user named Bob. This prevents the user from signing-in to the AWS console. It does not prevent the user from running any AWS CLI, PowerShell, or API calls using AWS access keys that might still be attached to the user account.

```
Remove-IAMLoginProfile -UserName Bob
```

- For API details, see DeleteLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the login profile from the IAM user named Bob. This prevents the user from signing-in to the AWS console. It does not prevent the user from running any AWS CLI, PowerShell, or API calls using AWS access keys that might still be attached to the user account.

```
Remove-IAMLoginProfile -UserName Bob
```

- For API details, see DeleteLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteOpenIdConnectProvider` with a CLI

The following code examples show how to use `DeleteOpenIdConnectProvider`.

CLI

AWS CLI

To delete an IAM OpenID Connect identity provider

The following `delete-open-id-connect-provider` example deletes the IAM OIDC provider that connects to the provider `example.oidcprovider.com`.

```
aws iam delete-open-id-connect-provider \
    --open-id-connect-provider-arn arn:aws:iam::123456789012:oidc-provider/
example.oidcprovider.com
```

This command produces no output.

For more information, see Creating OpenID Connect (OIDC) identity providers in the *AWS IAM User Guide*.

- For API details, see DeleteOpenIdConnectProvider in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the IAM OIDC provider that connects to the provider `example.oidcprovider.com`. Ensure that you update or delete any roles that reference this provider in the `Principal` element of the role's trust policy.

```
Remove-IAMOpenIDConnectProvider -OpenIDConnectProviderArn
 arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com
```

- For API details, see DeleteOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the IAM OIDC provider that connects to the provider `example.oidcprovider.com`. Ensure that you update or delete any roles that reference this provider in the `Principal` element of the role's trust policy.

```
Remove-IAMOpenIDConnectProvider -OpenIDConnectProviderArn
 arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com
```

- For API details, see DeleteOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeletePolicy` with an AWS SDK or CLI

The following code examples show how to use `DeletePolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Create read-only and read-write users
- Creating a managed monitoring workspace
- Getting started with managed streaming
- Getting started with workflow orchestration
- Manage policies
- Setting up systems management

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new
DeletePolicyRequest { PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DeletePolicy in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
###############################################################################
# function iecho
#
```

```
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function iam_delete_policy
#
# This function deletes an IAM policy.
#
# Parameters:
#       -n policy_arn -- The name of the IAM policy arn.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_policy() {
  local policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_policy"
    echo "Deletes an AWS Identity and Access Management (IAM) policy"
    echo "  -n policy_arn -- The name of the IAM policy arn."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
```

```
    case "${option}" in
      n) policy_arn="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$policy_arn" ]]; then
    errecho "ERROR: You must provide a policy arn with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    Policy arn:  $policy_arn"
  iecho ""

  response=$(aws iam delete-policy \
    --policy-arn "$policy_arn")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-policy operation failed.\n$response"
    return 1
  fi

  iecho "delete-policy response:$response"
  iecho

  return 0
}
```

- For API details, see DeletePolicy in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::deletePolicy(const Aws::String &policyArn,
                               const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::DeletePolicyRequest request;
    request.SetPolicyArn(policyArn);

    auto outcome = iam.DeletePolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error deleting policy with arn " << policyArn << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully deleted policy with arn " << policyArn
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see DeletePolicy in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an IAM policy

The following `delete-policy` example deletes the policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
aws iam delete-policy \
    --policy-arn arn:aws:iam::123456789012:policy/MySamplePolicy
```

This command produces no output.

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see DeletePolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// DeletePolicy deletes a policy.
```

```
func (wrapper PolicyWrapper) DeletePolicy(ctx context.Context, policyArn string)
 error {
 _, err := wrapper.IamClient.DeletePolicy(ctx, &iam.DeletePolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
 }
 return err
}
```

- For API details, see DeletePolicy in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.DeletePolicyRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeletePolicy {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <policyARN>\s

                Where:
                    policyARN - A policy ARN value to delete.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String policyARN = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        deleteIAMPolicy(iam, policyARN);
        iam.close();
    }

    public static void deleteIAMPolicy(IamClient iam, String policyARN) {
        try {
            DeletePolicyRequest request = DeletePolicyRequest.builder()
                    .policyArn(policyARN)
                    .build();

            iam.deletePolicy(request);
            System.out.println("Successfully deleted the policy");

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        System.out.println("Done");
    }
}
```

- For API details, see DeletePolicy in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete the policy.

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- For API details, see DeletePolicy in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteIAMPolicy(policyARNVal: String?) {
    val request =
```

```
        DeletePolicyRequest {
            policyArn = policyARNVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deletePolicy(request)
        println("Successfully deleted $policyARNVal")
    }
}
```

- For API details, see DeletePolicy in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`. Before you can delete the policy, you must first delete all versions except the default by running `Remove-IAMPolicyVersion`. You must also detach the policy from any IAM users, groups, or roles.

```
Remove-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
```

Example 2: This example deletes a policy by first deleting all the non-default policy versions, detaching it from all attached IAM entities, and finally deleting the policy itself. The first line retrieves the policy object. The second line retrieves all the policy versions that are not flagged as the default version into a collection and then deletes each policy in the collection. The third line retrieves all of the IAM users, groups, and roles to which the policy is attached. Lines four through six detach the policy from each attached entity. The last line uses this command to remove the managed policy as well as the remaining default version. The example includes the `-Force` switch parameter on any line that needs it to suppress prompts for confirmation.

```
$pol = Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
Get-IAMPolicyVersions -PolicyArn $pol.Arn | where {-not $_.IsDefaultVersion} |
 Remove-IAMPolicyVersion -PolicyArn $pol.Arn -force
$attached = Get-IAMEntitiesForPolicy -PolicyArn $pol.Arn
```

```
$attached.PolicyGroups | Unregister-IAMGroupPolicy -PolicyArn $pol.arn
$attached.PolicyRoles | Unregister-IAMRolePolicy -PolicyArn $pol.arn
$attached.PolicyUsers | Unregister-IAMUserPolicy -PolicyArn $pol.arn
Remove-IAMPolicy $pol.Arn -Force
```

- For API details, see DeletePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`. Before you can delete the policy, you must first delete all versions except the default by running `Remove-IAMPolicyVersion`. You must also detach the policy from any IAM users, groups, or roles.

```
Remove-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
```

Example 2: This example deletes a policy by first deleting all the non-default policy versions, detaching it from all attached IAM entities, and finally deleting the policy itself. The first line retrieves the policy object. The second line retrieves all the policy versions that are not flagged as the default version into a collection and then deletes each policy in the collection. The third line retrieves all of the IAM users, groups, and roles to which the policy is attached. Lines four through six detach the policy from each attached entity. The last line uses this command to remove the managed policy as well as the remaining default version. The example includes the `-Force` switch parameter on any line that needs it to suppress prompts for confirmation.

```
$pol = Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
Get-IAMPolicyVersions -PolicyArn $pol.Arn | where {-not $_.IsDefaultVersion} |
 Remove-IAMPolicyVersion -PolicyArn $pol.Arn -force
$attached = Get-IAMEntitiesForPolicy -PolicyArn $pol.Arn
$attached.PolicyGroups | Unregister-IAMGroupPolicy -PolicyArn $pol.arn
$attached.PolicyRoles | Unregister-IAMRolePolicy -PolicyArn $pol.arn
$attached.PolicyUsers | Unregister-IAMUserPolicy -PolicyArn $pol.arn
Remove-IAMPolicy $pol.Arn -Force
```

- For API details, see DeletePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def delete_policy(policy_arn):
    """
    Deletes a policy.

    :param policy_arn: The ARN of the policy to delete.
    """
    try:
        iam.Policy(policy_arn).delete()
        logger.info("Deleted policy %s.", policy_arn)
    except ClientError:
        logger.exception("Couldn't delete policy %s.", policy_arn)
        raise
```

- For API details, see DeletePolicy in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn delete_policy(client: &iamClient, policy: Policy) -> Result<(),
 iamError> {
```

```
    client
        .delete_policy()
        .policy_arn(policy.arn.unwrap())
        .send()
        .await?;
    Ok(())
}
```

- For API details, see DeletePolicy in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->deletepolicy( iv_policyarn = iv_policy_arn ).
        MESSAGE 'Policy deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Policy does not exist.' TYPE 'E'.
      CATCH /aws1/cx_iamdeleteconflictex.
        MESSAGE 'Policy cannot be deleted due to attachments.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeletePolicy in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func deletePolicy(policy: IAMClientTypes.Policy) async throws {
        let input = DeletePolicyInput(
            policyArn: policy.arn
        )
        do {
            _ = try await iamClient.deletePolicy(input: input)
        } catch {
            print("ERROR: deletePolicy:", dump(error))
            throw error
        }
    }
```

- For API details, see DeletePolicy in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeletePolicyVersion` with an AWS SDK or CLI

The following code examples show how to use `DeletePolicyVersion`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Manage policies
- Roll back a policy version

CLI

AWS CLI

To delete a version of a managed policy

The following `delete-policy-version` example deletes the version identified as `v2` from the policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy`.

```
aws iam delete-policy-version \
    --policy-arn arn:aws:iam::123456789012:policy/MyPolicy \
    --version-id v2
```

This command produces no output.

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see DeletePolicyVersion in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the version identified as v2 from the policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
Remove-IAMPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/
MySamplePolicy -VersionID v2
```

Example 2: This example deletes a policy by first deleting all non-default policy versions and then deleting the policy itself. The first line retrieves the policy object. The second line retrieves all of the policy versions that are not flagged as the default into a collection and then uses this command to delete each policy in the collection. The last line removes the policy itself as well as the remaining default version. Note that to successfully delete a managed policy, you must also detach the policy from any users, groups, or roles by using the `Unregister-IAMUserPolicy`, `Unregister-`

IAMGroupPolicy, and Unregister-IAMRolePolicy commands. See the example for the Remove-IAMPolicy cmdlet.

```
$pol = Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
Get-IAMPolicyVersions -PolicyArn $pol.Arn | where {-not $_.IsDefaultVersion} |
 Remove-IAMPolicyVersion -PolicyArn $pol.Arn -force
Remove-IAMPolicy -PolicyArn $pol.Arn -force
```

- For API details, see DeletePolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the version identified as v2 from the policy whose ARN is arn:aws:iam::123456789012:policy/MySamplePolicy.

```
Remove-IAMPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/
MySamplePolicy -VersionID v2
```

Example 2: This example deletes a policy by first deleting all non-default policy versions and then deleting the policy itself. The first line retrieves the policy object. The second line retrieves all of the policy versions that are not flagged as the default into a collection and then uses this command to delete each policy in the collection. The last line removes the policy itself as well as the remaining default version. Note that to successfully delete a managed policy, you must also detach the policy from any users, groups, or roles by using the Unregister-IAMUserPolicy, Unregister-IAMGroupPolicy, and Unregister-IAMRolePolicy commands. See the example for the Remove-IAMPolicy cmdlet.

```
$pol = Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
Get-IAMPolicyVersions -PolicyArn $pol.Arn | where {-not $_.IsDefaultVersion} |
 Remove-IAMPolicyVersion -PolicyArn $pol.Arn -force
Remove-IAMPolicy -PolicyArn $pol.Arn -force
```

- For API details, see DeletePolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_iam->deletepolicyversion(
      iv_policyarn = iv_policy_arn
      iv_versionid = iv_version_id ).
    MESSAGE 'Policy version deleted successfully.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'Policy or version does not exist.' TYPE 'E'.
  CATCH /aws1/cx_iamdeleteconflictex.
    MESSAGE 'Cannot delete default policy version.' TYPE 'E'.
  CATCH /aws1/cx_iamlimitexceededex.
    MESSAGE 'Limit exceeded.' TYPE 'E'.
ENDTRY.
```

- For API details, see DeletePolicyVersion in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteRole` with an AWS SDK or CLI

The following code examples show how to use `DeleteRole`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Configure container service connectivity
- Create a rest API with function proxy integration

- Creating a managed monitoring workspace
- Creating a monitoring dashboard with function name as a variable
- Creating your first serverless function
- Get started with serverless data warehouses
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with provisioned data warehouse clusters
- Getting started with workflow orchestration
- Manage roles
- Moving hardcoded secrets to secure secret storage
- Run CPU stress tests on virtual machine instances using fault injection
- Setting up systems management
- Using property variables in monitoring dashboards to monitor multiple serverless functions

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Delete an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRoleAsync(string roleName)
{
```

```
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
 { RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DeleteRole in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
##############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function iam_delete_role
```

```
#
# This function deletes an IAM role.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_role() {
  local role_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_role"
    echo "Deletes an AWS Identity and Access Management (IAM) role"
    echo "  -n role_name -- The name of the IAM role."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) role_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  echo "role_name:$role_name"
  if [[ -z "$role_name" ]]; then
    errecho "ERROR: You must provide a role name with the -n parameter."
    usage
    return 1
```

```
  fi

  iecho "Parameters:\n"
  iecho "    Role name:  $role_name"
  iecho ""

  response=$(aws iam delete-role \
    --role-name "$role_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-role operation failed.\n$response"
    return 1
  fi

  iecho "delete-role response:$response"
  iecho

  return 0
}
```

- For API details, see DeleteRole in *AWS CLI Command Reference*.

CLI

AWS CLI

To delete an IAM role

The following `delete-role` example removes the role named `Test-Role`.

```
aws iam delete-role \
    --role-name Test-Role
```

This command produces no output.

Before you can delete a role, you must remove the role from any instance profile (`remove-role-from-instance-profile`), detach any managed policies (`detach-role-policy`) and delete any inline policies that are attached to the role (`delete-role-policy`).

For more information, see Creating IAM roles and Using instance profiles in the *AWS IAM User Guide*.

- For API details, see DeleteRole in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(ctx context.Context, roleName string) error
 {
 _, err := wrapper.IamClient.DeleteRole(ctx, &iam.DeleteRoleInput{
  RoleName: aws.String(roleName),
 })
```

```
 if err != nil {
  log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

- For API details, see DeleteRole in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete the role.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- For API details, see DeleteRole in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the role named `MyNewRole` from the current IAM account. Before you can delete the role you must first use the `Unregister-IAMRolePolicy` command to detach any managed policies. Inline policies are deleted with the role.

```
Remove-IAMRole -RoleName MyNewRole
```

Example 2: This example detaches any managed policies from the role named `MyNewRole` and then deletes the role. The first line retrieves any managed policies attached to the role as a collection and then detaches each policy in the collection from the role. The second line deletes the role itself. Inline policies are deleted along with the role.

```
Get-IAMAttachedRolePolicyList -RoleName MyNewRole | Unregister-IAMRolePolicy -
RoleName MyNewRole
Remove-IAMRole -RoleName MyNewRole
```

- For API details, see DeleteRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the role named `MyNewRole` from the current IAM account. Before you can delete the role you must first use the `Unregister-IAMRolePolicy` command to detach any managed policies. Inline policies are deleted with the role.

```
Remove-IAMRole -RoleName MyNewRole
```

Example 2: This example detaches any managed policies from the role named `MyNewRole` and then deletes the role. The first line retrieves any managed policies attached to the role as a collection and then detaches each policy in the collection from the role. The second line deletes the role itself. Inline policies are deleted along with the role.

```
Get-IAMAttachedRolePolicyList -RoleName MyNewRole | Unregister-IAMRolePolicy -
RoleName MyNewRole
```

```
Remove-IAMRole -RoleName MyNewRole
```

- For API details, see DeleteRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def delete_role(role_name):
    """
    Deletes a role.

    :param role_name: The name of the role to delete.
    """
    try:
        iam.Role(role_name).delete()
        logger.info("Deleted role %s.", role_name)
    except ClientError:
        logger.exception("Couldn't delete role %s.", role_name)
        raise
```

- For API details, see DeleteRole in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
  # Deletes a role and its attached policies.
  #
  # @param role_name [String] The name of the role to delete.
  def delete_role(role_name)
    # Detach and delete attached policies
    @iam_client.list_attached_role_policies(role_name: role_name).each do |
response|
      response.attached_policies.each do |policy|
        @iam_client.detach_role_policy({
                                         role_name: role_name,
                                         policy_arn: policy.policy_arn
                                       })
        # Check if the policy is a customer managed policy (not AWS managed)
        unless policy.policy_arn.include?('aws:policy/')
          @iam_client.delete_policy({ policy_arn: policy.policy_arn })
          @logger.info("Deleted customer managed policy #{policy.policy_name}.")
        end
      end
    end

    # Delete the role
    @iam_client.delete_role({ role_name: role_name })
    @logger.info("Deleted role #{role_name}.")
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't detach policies and delete role #{role_name}. Here's
 why:")
    @logger.error("\t#{e.code}: #{e.message}")
    raise
  end
```

- For API details, see DeleteRole in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn delete_role(client: &iamClient, role: &Role) -> Result<(), iamError>
 {
    let role = role.clone();
    while client
        .delete_role()
        .role_name(role.role_name())
        .send()
        .await
        .is_err()
    {
        sleep(Duration::from_secs(2)).await;
    }
    Ok(())
}
```

- For API details, see DeleteRole in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->deleterole( iv_rolename = iv_role_name ).
```

```
        MESSAGE 'Role deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Role does not exist.' TYPE 'E'.
      CATCH /aws1/cx_iamdeleteconflictex.
        MESSAGE 'Role cannot be deleted due to attached resources.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeleteRole in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func deleteRole(role: IAMClientTypes.Role) async throws {
        let input = DeleteRoleInput(
            roleName: role.roleName
        )
        do {
            _ = try await iamClient.deleteRole(input: input)
        } catch {
            print("ERROR: deleteRole:", dump(error))
            throw error
        }
    }
```

- For API details, see DeleteRole in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteRolePermissionsBoundary` with a CLI

The following code examples show how to use `DeleteRolePermissionsBoundary`.

CLI

AWS CLI

To delete a permissions boundary from an IAM role

The following `delete-role-permissions-boundary` example deletes the permissions boundary for the specified IAM role. To apply a permissions boundary to a role, use the `put-role-permissions-boundary` command.

```
aws iam delete-role-permissions-boundary \
    --role-name lambda-application-role
```

This command produces no output.

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see DeleteRolePermissionsBoundary in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example shows how to remove the permission boundary attached to an IAM role.

```
Remove-IAMRolePermissionsBoundary -RoleName MyRoleName
```

- For API details, see DeleteRolePermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example shows how to remove the permission boundary attached to an IAM role.

```
Remove-IAMRolePermissionsBoundary -RoleName MyRoleName
```

- For API details, see DeleteRolePermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteRolePolicy` with an AWS SDK or CLI

The following code examples show how to use `DeleteRolePolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Get started with serverless data warehouses
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with provisioned data warehouse clusters
- Run CPU stress tests on virtual machine instances using fault injection

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an IAM role policy.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</
param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DeleteRolePolicy in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To remove a policy from an IAM role

The following `delete-role-policy` example removes the policy named `ExamplePolicy` from the role named `Test-Role`.

```
aws iam delete-role-policy \
    --role-name Test-Role \
    --policy-name ExamplePolicy
```

This command produces no output.

For more information, see IAM role management in the *AWS IAM User Guide*.

- For API details, see DeleteRolePolicy in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
  return client.send(command);
};
```

- For API details, see DeleteRolePolicy in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the inline policy `S3AccessPolicy` that is embedded in the IAM role `S3BackupRole`.

```
Remove-IAMRolePolicy -PolicyName S3AccessPolicy -RoleName S3BackupRole
```

- For API details, see DeleteRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the inline policy `S3AccessPolicy` that is embedded in the IAM role `S3BackupRole`.

```
Remove-IAMRolePolicy -PolicyName S3AccessPolicy -RoleName S3BackupRole
```

- For API details, see DeleteRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteSAMLProvider` with an AWS SDK or CLI

The following code examples show how to use `DeleteSAMLProvider`.

CLI

AWS CLI

To delete a SAML provider

The following `delete-saml-provider` example deletes the IAM SAML 2.0 provider whose ARN is `arn:aws:iam::123456789012:saml-provider/SAMLADFSProvider`.

```
aws iam delete-saml-provider \
    --saml-provider-arn arn:aws:iam::123456789012:saml-provider/SAMLADFSProvider
```

This command produces no output.

For more information, see Creating IAM SAML identity providers in the *AWS IAM User Guide*.

- For API details, see DeleteSAMLProvider in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For API details, see DeleteSAMLProvider in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the IAM SAML 2.0 provider whose ARN is `arn:aws:iam::123456789012:saml-provider/SAMLADFSProvider`.

```
Remove-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/
SAMLADFSProvider
```

- For API details, see DeleteSAMLProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the IAM SAML 2.0 provider whose ARN is `arn:aws:iam::123456789012:saml-provider/SAMLADFSProvider`.

```
Remove-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/
SAMLADFSProvider
```

- For API details, see DeleteSAMLProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteServerCertificate` with an AWS SDK or CLI

The following code examples show how to use `DeleteServerCertificate`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::deleteServerCertificate(const Aws::String &certificateName,
                                          const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::DeleteServerCertificateRequest request;
    request.SetServerCertificateName(certificateName);

    const auto outcome = iam.DeleteServerCertificate(request);
    bool result = true;
```

```
    if (!outcome.IsSuccess()) {
        if (outcome.GetError().GetErrorType() !=
 Aws::IAM::IAMErrors::NO_SUCH_ENTITY) {
            std::cerr << "Error deleting server certificate " << certificateName
 <<
                      ": " << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Certificate '" << certificateName
                      << "' not found." << std::endl;
        }
    }
    else {
        std::cout << "Successfully deleted server certificate " <<
 certificateName
                  << std::endl;
    }

    return result;
}
```

- For API details, see DeleteServerCertificate in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete a server certificate from your AWS account

The following `delete-server-certificate` example removes the specified server certificate from your AWS account.

```
aws iam delete-server-certificate \
    --server-certificate-name myUpdatedServerCertificate
```

This command produces no output.

To list the server certificates available in your AWS account, use the `list-server-certificates` command.

For more information, see Managing server certificates in IAM in the *AWS IAM User Guide*.

- For API details, see DeleteServerCertificate in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete a server certificate.

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
  const command = new DeleteServerCertificateCommand({
    ServerCertificateName: certName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteServerCertificate in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteServerCertificate in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the server certificate named `MyServerCert`.

```
Remove-IAMServerCertificate -ServerCertificateName MyServerCert
```

- For API details, see DeleteServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the server certificate named `MyServerCert`.

```
Remove-IAMServerCertificate -ServerCertificateName MyServerCert
```

- For API details, see DeleteServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List, update, and delete server certificates.

```
class ServerCertificateManager
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'ServerCertificateManager'
  end

  # Creates a new server certificate.
  # @param name [String] the name of the server certificate
  # @param certificate_body [String] the contents of the certificate
  # @param private_key [String] the private key contents
  # @return [Boolean] returns true if the certificate was successfully created
  def create_server_certificate(name, certificate_body, private_key)
    @iam_client.upload_server_certificate({
                                            server_certificate_name: name,
                                            certificate_body: certificate_body,
                                            private_key: private_key
                                          })
    true
  rescue Aws::IAM::Errors::ServiceError => e
    puts "Failed to create server certificate: #{e.message}"
    false
  end

  # Lists available server certificate names.
  def list_server_certificate_names
    response = @iam_client.list_server_certificates

    if response.server_certificate_metadata_list.empty?
      @logger.info('No server certificates found.')
      return
```

```
    end

    response.server_certificate_metadata_list.each do |certificate_metadata|
      @logger.info("Certificate Name:
 #{certificate_metadata.server_certificate_name}")
    end
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing server certificates: #{e.message}")
  end

  # Updates the name of a server certificate.
  def update_server_certificate_name(current_name, new_name)
    @iam_client.update_server_certificate(
      server_certificate_name: current_name,
      new_server_certificate_name: new_name
    )
    @logger.info("Server certificate name updated from '#{current_name}' to
 '#{new_name}'.")
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error updating server certificate name: #{e.message}")
    false
  end

  # Deletes a server certificate.
  def delete_server_certificate(name)
    @iam_client.delete_server_certificate(server_certificate_name: name)
    @logger.info("Server certificate '#{name}' deleted.")
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting server certificate: #{e.message}")
    false
  end
end
```

- For API details, see DeleteServerCertificate in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteServiceLinkedRole` with an AWS SDK or CLI

The following code examples show how to use `DeleteServiceLinkedRole`.

CLI

AWS CLI

To delete a service-linked role

The following `delete-service-linked-role` example deletes the specified service-linked role that you no longer need. The deletion happens asynchronously. You can check the status of the deletion and confirm when it is done by using the `get-service-linked-role-deletion-status` command.

```
aws iam delete-service-linked-role \
    --role-name AWSServiceRoleForLexBots
```

Output:

```
{
    "DeletionTaskId": "task/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

For more information, see Using service-linked roles in the *AWS IAM User Guide*.

- For API details, see DeleteServiceLinkedRole in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(ctx context.Context, roleName
 string) error {
 _, err := wrapper.IamClient.DeleteServiceLinkedRole(ctx,
 &iam.DeleteServiceLinkedRoleInput{
  RoleName: aws.String(roleName)},
 )
 if err != nil {
  log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n",
 roleName, err)
 }
 return err
}
```

- For API details, see DeleteServiceLinkedRole in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- For API details, see DeleteServiceLinkedRole in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deleted the service linked role. Please note that if the service is still using this role, then this command results in a failure.

```
Remove-IAMServiceLinkedRole -RoleName
 AWSServiceRoleForAutoScaling_RoleNameEndsWithThis
```

- For API details, see DeleteServiceLinkedRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deleted the service linked role. Please note that if the service is still using this role, then this command results in a failure.

```
Remove-IAMServiceLinkedRole -RoleName
 AWSServiceRoleForAutoScaling_RoleNameEndsWithThis
```

- For API details, see DeleteServiceLinkedRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Deletes a service-linked role.
#
# @param role_name [String] The name of the role to delete.
def delete_service_linked_role(role_name)
  response = @iam_client.delete_service_linked_role(role_name: role_name)
  task_id = response.deletion_task_id
  check_deletion_status(role_name, task_id)
rescue Aws::Errors::ServiceError => e
  handle_deletion_error(e, role_name)
end

private

# Checks the deletion status of a service-linked role
#
# @param role_name [String] The name of the role being deleted
# @param task_id [String] The task ID for the deletion process
def check_deletion_status(role_name, task_id)
  loop do
    response = @iam_client.get_service_linked_role_deletion_status(
```

```
        deletion_task_id: task_id
      )
      status = response.status
      @logger.info("Deletion of #{role_name} #{status}.")
      break if %w[SUCCEEDED FAILED].include?(status)

      sleep(3)
    end
  end

  # Handles deletion error
  #
  # @param e [Aws::Errors::ServiceError] The error encountered during deletion
  # @param role_name [String] The name of the role attempted to delete
  def handle_deletion_error(e, role_name)
    return if e.code == 'NoSuchEntity'

    @logger.error("Couldn't delete #{role_name}. Here's why:")
    @logger.error("\t#{e.code}: #{e.message}")
    raise
  end
```

- For API details, see DeleteServiceLinkedRole in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn delete_service_linked_role(
    client: &iamClient,
    role_name: &str,
) -> Result<(), iamError> {
    client
        .delete_service_linked_role()
        .role_name(role_name)
```

```
        .send()
        .await?;

    Ok(())
}
```

- For API details, see DeleteServiceLinkedRole in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteSigningCertificate` with a CLI

The following code examples show how to use `DeleteSigningCertificate`.

CLI

AWS CLI

To delete a signing certificate for an IAM user

The following `delete-signing-certificate` example deletes the specified signing certificate for the IAM user named Bob.

```
aws iam delete-signing-certificate \
    --user-name Bob \
    --certificate-id TA7SMP42TDN5Z26OBPJE7EXAMPLE
```

This command produces no output.

To get the ID for a signing certificate, use the `list-signing-certificates` command.

For more information, see Manage signing certificates in the *Amazon EC2 User Guide*.

- For API details, see DeleteSigningCertificate in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the signing certificate with the ID `Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU` from the IAM user named Bob.

```
Remove-IAMSigningCertificate -UserName Bob -CertificateId
 Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU
```

- For API details, see DeleteSigningCertificate in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the signing certificate with the ID `Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU` from the IAM user named Bob.

```
Remove-IAMSigningCertificate -UserName Bob -CertificateId
 Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU
```

- For API details, see DeleteSigningCertificate in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteUser` with an AWS SDK or CLI

The following code examples show how to use `DeleteUser`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Create read-only and read-write users

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DeleteUser in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function iecho
#
```

```
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function iam_delete_user
#
# This function deletes the specified IAM user.
#
# Parameters:
#       -u user_name  -- The name of the user to create.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function iam_delete_user() {
  local user_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_delete_user"
    echo "Deletes an AWS Identity and Access Management (IAM) user. You must
 supply a username:"
    echo "  -u user_name    The name of the user."
    echo ""
  }

  # Retrieve the calling parameters.
```

```
while getopts "u:h" option; do
  case "${option}" in
    u) user_name="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$user_name" ]]; then
  errecho "ERROR: You must provide a username with the -u parameter."
  usage
  return 1
fi

iecho "Parameters:\n"
iecho "    User name:   $user_name"
iecho ""

# If the user does not exist, we don't want to try to delete it.
if (! iam_user_exists "$user_name"); then
  errecho "ERROR: A user with that name does not exist in the account."
  return 1
fi

response=$(aws iam delete-user \
  --user-name "$user_name")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports delete-user operation failed.$response"
  return 1
fi

iecho "delete-user response:$response"
```

```
  iecho

  return 0
}
```

- For API details, see DeleteUser in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::DeleteUserRequest request;
    request.SetUserName(userName);
    auto outcome = iam.DeleteUser(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error deleting IAM user " << userName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;;
    }
    else {
        std::cout << "Successfully deleted IAM user " << userName << std::endl;
    }

    return outcome.IsSuccess();
```

- For API details, see DeleteUser in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an IAM user

The following `delete-user` example removes the IAM user named Bob from the current account.

```
aws iam delete-user \
    --user-name Bob
```

This command produces no output.

For more information, see Deleting an IAM user in the *AWS IAM User Guide*.

- For API details, see DeleteUser in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}
```

```
// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(ctx context.Context, userName string) error
 {
 _, err := wrapper.IamClient.DeleteUser(ctx, &iam.DeleteUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

- For API details, see DeleteUser in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.DeleteUserRequest;
import software.amazon.awssdk.services.iam.model.IamException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteUser {
```

```
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <userName>\s

                Where:
                    userName - The name of the user to delete.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String userName = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        deleteIAMUser(iam, userName);
        System.out.println("Done");
        iam.close();
    }

    public static void deleteIAMUser(IamClient iam, String userName) {
        try {
            DeleteUserRequest request = DeleteUserRequest.builder()
                    .userName(userName)
                    .build();

            iam.deleteUser(request);
            System.out.println("Successfully deleted IAM user " + userName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteUser in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete the user.

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteUser in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteUser in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteIAMUser(userNameVal: String) {
    val request =
        DeleteUserRequest {
            userName = userNameVal
```

```
        }

    // To delete a user, ensure that the user's access keys are deleted first.
    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.deleteUser(request)
        println("Successfully deleted user $userNameVal")
    }
}
```

- For API details, see DeleteUser in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the IAM user named Bob.

```
Remove-IAMUser -UserName Bob
```

Example 2: This example deletes the IAM user named Theresa along with any elements that must be deleted first.

```
$name = "Theresa"

# find any groups and remove user from them
$groups = Get-IAMGroupForUser -UserName $name
foreach ($group in $groups) { Remove-IAMUserFromGroup -GroupName $group.GroupName
 -UserName $name -Force }

# find any inline policies and delete them
$inlinepols = Get-IAMUserPolicies -UserName $name
foreach ($pol in $inlinepols) { Remove-IAMUserPolicy -PolicyName $pol -UserName
 $name -Force}

# find any managed polices and detach them
$managedpols = Get-IAMAttachedUserPolicies -UserName $name
foreach ($pol in $managedpols) { Unregister-IAMUserPolicy -PolicyArn
 $pol.PolicyArn -UserName $name }

# find any signing certificates and delete them
$certs = Get-IAMSigningCertificate -UserName $name
```

```
foreach ($cert in $certs) { Remove-IAMSigningCertificate -CertificateId
 $cert.CertificateId -UserName $name -Force }

# find any access keys and delete them
$keys = Get-IAMAccessKey -UserName $name
foreach ($key in $keys) { Remove-IAMAccessKey -AccessKeyId $key.AccessKeyId -
UserName $name -Force }

# delete the user's login profile, if one exists - note: need to use try/catch to
 suppress not found error
try { $prof = Get-IAMLoginProfile -UserName $name -ea 0 } catch { out-null }
if ($prof) { Remove-IAMLoginProfile -UserName $name -Force }

# find any MFA device, detach it, and if virtual, delete it.
$mfa = Get-IAMMFADevice -UserName $name
if ($mfa) {
    Disable-IAMMFADevice -SerialNumber $mfa.SerialNumber -UserName $name
    if ($mfa.SerialNumber -like "arn:*") { Remove-IAMVirtualMFADevice -
SerialNumber $mfa.SerialNumber }
}

# finally, remove the user
Remove-IAMUser -UserName $name -Force
```

- For API details, see DeleteUser in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the IAM user named Bob.

```
Remove-IAMUser -UserName Bob
```

Example 2: This example deletes the IAM user named Theresa along with any elements that must be deleted first.

```
$name = "Theresa"

# find any groups and remove user from them
$groups = Get-IAMGroupForUser -UserName $name
foreach ($group in $groups) { Remove-IAMUserFromGroup -GroupName $group.GroupName
 -UserName $name -Force }

# find any inline policies and delete them
```

```
$inlinepols = Get-IAMUserPolicies -UserName $name
foreach ($pol in $inlinepols) { Remove-IAMUserPolicy -PolicyName $pol -UserName
 $name -Force}

# find any managed polices and detach them
$managedpols = Get-IAMAttachedUserPolicies -UserName $name
foreach ($pol in $managedpols) { Unregister-IAMUserPolicy -PolicyArn
 $pol.PolicyArn -UserName $name }

# find any signing certificates and delete them
$certs = Get-IAMSigningCertificate -UserName $name
foreach ($cert in $certs) { Remove-IAMSigningCertificate -CertificateId
 $cert.CertificateId -UserName $name -Force }

# find any access keys and delete them
$keys = Get-IAMAccessKey -UserName $name
foreach ($key in $keys) { Remove-IAMAccessKey -AccessKeyId $key.AccessKeyId -
UserName $name -Force }

# delete the user's login profile, if one exists - note: need to use try/catch to
 suppress not found error
try { $prof = Get-IAMLoginProfile -UserName $name -ea 0 } catch { out-null }
if ($prof) { Remove-IAMLoginProfile -UserName $name -Force }

# find any MFA device, detach it, and if virtual, delete it.
$mfa = Get-IAMMFADevice -UserName $name
if ($mfa) {
    Disable-IAMMFADevice -SerialNumber $mfa.SerialNumber -UserName $name
    if ($mfa.SerialNumber -like "arn:*") { Remove-IAMVirtualMFADevice -
SerialNumber $mfa.SerialNumber }
}

# finally, remove the user
Remove-IAMUser -UserName $name -Force
```

- For API details, see DeleteUser in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def delete_user(user_name):
    """
    Deletes a user. Before a user can be deleted, all associated resources,
    such as access keys and policies, must be deleted or detached.

    :param user_name: The name of the user.
    """
    try:
        iam.User(user_name).delete()
        logger.info("Deleted user %s.", user_name)
    except ClientError:
        logger.exception("Couldn't delete user %s.", user_name)
        raise
```

- For API details, see DeleteUser in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
  # Deletes a user and their associated resources
```

```
  #
  # @param user_name [String] The name of the user to delete
  def delete_user(user_name)
    user = @iam_client.list_access_keys(user_name: user_name).access_key_metadata
    user.each do |key|
      @iam_client.delete_access_key({ access_key_id: key.access_key_id,
 user_name: user_name })
      @logger.info("Deleted access key #{key.access_key_id} for user
 '#{user_name}'.")
    end

    @iam_client.delete_user(user_name: user_name)
    @logger.info("Deleted user '#{user_name}'.")
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting user '#{user_name}': #{e.message}")
  end
```

- For API details, see DeleteUser in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn delete_user(client: &iamClient, user: &User) -> Result<(),
 SdkError<DeleteUserError>> {
    let user = user.clone();
    let mut tries: i32 = 0;
    let max_tries: i32 = 10;

    let response: Result<(), SdkError<DeleteUserError>> = loop {
        match client
            .delete_user()
            .user_name(user.user_name())
            .send()
            .await
```

```
        {
            Ok(_) => {
                break Ok(());
            }
            Err(e) => {
                tries += 1;
                if tries > max_tries {
                    break Err(e);
                }
                sleep(Duration::from_secs(2)).await;
            }
        }
    };

    response
}
```

- For API details, see DeleteUser in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->deleteuser( iv_username = iv_user_name ).
        MESSAGE 'User deleted successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'User does not exist.' TYPE 'E'.
      CATCH /aws1/cx_iamdeleteconflictex.
        MESSAGE 'User cannot be deleted due to attached resources.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DeleteUser in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func deleteUser(user: IAMClientTypes.User) async throws {
        let input = DeleteUserInput(
            userName: user.userName
        )
        do {
            _ = try await iamClient.deleteUser(input: input)
        } catch {
            print("ERROR: deleteUser:", dump(error))
            throw error
        }
    }
```

- For API details, see DeleteUser in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteUserPermissionsBoundary` with a CLI

The following code examples show how to use `DeleteUserPermissionsBoundary`.

CLI

AWS CLI

To delete a permissions boundary from an IAM user

The following `delete-user-permissions-boundary` example deletes the permissions boundary attached to the IAM user named `intern`. To apply a permissions boundary to a user, use the `put-user-permissions-boundary` command.

```
aws iam delete-user-permissions-boundary \
    --user-name intern
```

This command produces no output.

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see DeleteUserPermissionsBoundary in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example shows how to remove the permission boundary attached to an IAM user.

```
Remove-IAMUserPermissionsBoundary -UserName joe
```

- For API details, see DeleteUserPermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example shows how to remove the permission boundary attached to an IAM user.

```
Remove-IAMUserPermissionsBoundary -UserName joe
```

- For API details, see DeleteUserPermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteUserPolicy` with an AWS SDK or CLI

The following code examples show how to use `DeleteUserPolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DeleteUserPolicy in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To remove a policy from an IAM user

The following `delete-user-policy` example removes the specified policy from the IAM user named Bob.

```
aws iam delete-user-policy \
    --user-name Bob \
    --policy-name ExamplePolicy
```

This command produces no output.

To get a list of policies for an IAM user, use the `list-user-policies` command.

For more information, see Creating an IAM user in your AWS account in the *AWS IAM User Guide*.

- For API details, see DeleteUserPolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(ctx context.Context, userName string,
 policyName string) error {
 _, err := wrapper.IamClient.DeleteUserPolicy(ctx, &iam.DeleteUserPolicyInput{
  PolicyName: aws.String(policyName),
  UserName:   aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName,
 err)
 }
 return err
}
```

- For API details, see DeleteUserPolicy in *AWS SDK for Go API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the inline policy named `AccessToEC2Policy` that is embedded in the IAM user named `Bob`.

```
Remove-IAMUserPolicy -PolicyName AccessToEC2Policy -UserName Bob
```

Example 2: This example finds all of the inline polices that are embedded in the IAM user named `Theresa` and then deletes them.

```
$inlinepols = Get-IAMUserPolicies -UserName Theresa
foreach ($pol in $inlinepols) { Remove-IAMUserPolicy -PolicyName $pol -UserName
 Theresa -Force}
```

- For API details, see DeleteUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the inline policy named `AccessToEC2Policy` that is embedded in the IAM user named Bob.

```
Remove-IAMUserPolicy -PolicyName AccessToEC2Policy -UserName Bob
```

Example 2: This example finds all of the inline polices that are embedded in the IAM user named `Theresa` and then deletes them.

```
$inlinepols = Get-IAMUserPolicies -UserName Theresa
foreach ($pol in $inlinepols) { Remove-IAMUserPolicy -PolicyName $pol -UserName
 Theresa -Force}
```

- For API details, see DeleteUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
  # Deletes a user and their associated resources
  #
  # @param user_name [String] The name of the user to delete
  def delete_user(user_name)
    user = @iam_client.list_access_keys(user_name: user_name).access_key_metadata
    user.each do |key|
      @iam_client.delete_access_key({ access_key_id: key.access_key_id,
user_name: user_name })
```

```
      @logger.info("Deleted access key #{key.access_key_id} for user
 '#{user_name}'.")
    end

    @iam_client.delete_user(user_name: user_name)
    @logger.info("Deleted user '#{user_name}'.")
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting user '#{user_name}': #{e.message}")
  end
```

- For API details, see DeleteUserPolicy in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn delete_user_policy(
    client: &iamClient,
    user: &User,
    policy_name: &str,
) -> Result<(), SdkError<DeleteUserPolicyError>> {
    client
        .delete_user_policy()
        .user_name(user.user_name())
        .policy_name(policy_name)
        .send()
        .await?;

    Ok(())
}
```

- For API details, see DeleteUserPolicy in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    func deleteUserPolicy(user: IAMClientTypes.User, policyName: String) async
 throws {
        let input = DeleteUserPolicyInput(
            policyName: policyName,
            userName: user.userName
        )
        do {
            _ = try await iamClient.deleteUserPolicy(input: input)
        } catch {
            print("ERROR: deleteUserPolicy:", dump(error))
            throw error
        }
    }
```

- For API details, see DeleteUserPolicy in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteVirtualMfaDevice` with a CLI

The following code examples show how to use `DeleteVirtualMfaDevice`.

CLI

AWS CLI

To remove a virtual MFA device

The following `delete-virtual-mfa-device` example removes the specified MFA device from the current account.

```
aws iam delete-virtual-mfa-device \
    --serial-number arn:aws:iam::123456789012:mfa/MFATest
```

This command produces no output.

For more information, see Deactivating MFA devices in the *AWS IAM User Guide*.

- For API details, see DeleteVirtualMfaDevice in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the IAM virtual MFA device whose ARN is `arn:aws:iam::123456789012:mfa/bob`.

```
Remove-IAMVirtualMFADevice -SerialNumber arn:aws:iam::123456789012:mfa/bob
```

Example 2: This example checks to see whether the IAM user Theresa has an MFA device assigned. If one is found, the device is disabled for the IAM user. If the device is virtual, then it is also deleted.

```
$mfa = Get-IAMMFADevice -UserName Theresa
if ($mfa) {
    Disable-IAMMFADevice -SerialNumber $mfa.SerialNumber -UserName $name
    if ($mfa.SerialNumber -like "arn:*") { Remove-IAMVirtualMFADevice -
SerialNumber $mfa.SerialNumber }
}
```

- For API details, see DeleteVirtualMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the IAM virtual MFA device whose ARN is `arn:aws:iam::123456789012:mfa/bob`.

```
Remove-IAMVirtualMFADevice -SerialNumber arn:aws:iam::123456789012:mfa/bob
```

Example 2: This example checks to see whether the IAM user Theresa has an MFA device assigned. If one is found, the device is disabled for the IAM user. If the device is virtual, then it is also deleted.

```
$mfa = Get-IAMMFADevice -UserName Theresa
if ($mfa) {
    Disable-IAMMFADevice -SerialNumber $mfa.SerialNumber -UserName $name
    if ($mfa.SerialNumber -like "arn:*") { Remove-IAMVirtualMFADevice -
SerialNumber $mfa.SerialNumber }
}
```

- For API details, see DeleteVirtualMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DetachGroupPolicy` with a CLI

The following code examples show how to use `DetachGroupPolicy`.

CLI

AWS CLI

To detach a policy from a group

The following `detach-group-policy` example removes the managed policy with the ARN `arn:aws:iam::123456789012:policy/TesterAccessPolicy` from the group called `Testers`.

```
aws iam detach-group-policy \
```

```
    --group-name Testers \
    --policy-arn arn:aws:iam::123456789012:policy/TesterAccessPolicy
```

This command produces no output.

For more information, see Managing IAM user groups in the *AWS IAM User Guide*.

- For API details, see DetachGroupPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example detaches the managed group policy whose ARN is `arn:aws:iam::123456789012:policy/TesterAccessPolicy` from the group named `Testers`.

```
Unregister-IAMGroupPolicy -GroupName Testers -PolicyArn
 arn:aws:iam::123456789012:policy/TesterAccessPolicy
```

Example 2: This example finds all the managed policies that are attached to the group named `Testers` and detaches them from the group.

```
Get-IAMAttachedGroupPolicies -GroupName Testers | Unregister-IAMGroupPolicy -
Groupname Testers
```

- For API details, see DetachGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example detaches the managed group policy whose ARN is `arn:aws:iam::123456789012:policy/TesterAccessPolicy` from the group named `Testers`.

```
Unregister-IAMGroupPolicy -GroupName Testers -PolicyArn
 arn:aws:iam::123456789012:policy/TesterAccessPolicy
```

Example 2: This example finds all the managed policies that are attached to the group named `Testers` and detaches them from the group.

```
Get-IAMAttachedGroupPolicies -GroupName Testers | Unregister-IAMGroupPolicy -
Groupname Testers
```

- For API details, see DetachGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DetachRolePolicy` with an AWS SDK or CLI

The following code examples show how to use `DetachRolePolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Configure container service connectivity
- Create a rest API with function proxy integration
- Creating a managed monitoring workspace
- Creating a monitoring dashboard with function name as a variable
- Creating your first serverless function
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with workflow orchestration
- Manage roles
- Moving hardcoded secrets to secure secret storage
- Run CPU stress tests on virtual machine instances using fault injection
- Setting up systems management
- Using property variables in monitoring dashboards to monitor multiple serverless functions

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM
policy.</param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DetachRolePolicy in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function iam_detach_role_policy
#
# This function detaches an IAM policy to a tole.
#
# Parameters:
#       -n role_name -- The name of the IAM role.
#       -p policy_ARN -- The IAM policy document ARN..
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function iam_detach_role_policy() {
  local role_name policy_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_detach_role_policy"
    echo "Detaches an AWS Identity and Access Management (IAM) policy to an IAM
 role."
    echo "  -n role_name   The name of the IAM role."
```

```
  echo "  -p policy_ARN -- The IAM policy document ARN."
  echo ""
}

# Retrieve the calling parameters.
while getopts "n:p:h" option; do
  case "${option}" in
    n) role_name="${OPTARG}" ;;
    p) policy_arn="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$role_name" ]]; then
  errecho "ERROR: You must provide a role name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$policy_arn" ]]; then
  errecho "ERROR: You must provide a policy ARN with the -p parameter."
  usage
  return 1
fi

response=$(aws iam detach-role-policy \
  --role-name "$role_name" \
  --policy-arn "$policy_arn")

local error_code=${?}

if [[ $error_code -ne 0 ]]; then
  aws_cli_error_log $error_code
  errecho "ERROR: AWS reports detach-role-policy operation failed.\n$response"
  return 1
```

```
  fi

  echo "$response"

  return 0
}
```

- For API details, see DetachRolePolicy in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::DetachRolePolicyRequest detachRequest;
    detachRequest.SetRoleName(roleName);
    detachRequest.SetPolicyArn(policyArn);

    auto detachOutcome = iam.DetachRolePolicy(detachRequest);
    if (!detachOutcome.IsSuccess()) {
        std::cerr << "Failed to detach policy " << policyArn << " from role "
                  << roleName << ": " << detachOutcome.GetError().GetMessage() <<
                  std::endl;
    }
    else {
        std::cout << "Successfully detached policy " << policyArn << " from role
 "
                  << roleName << std::endl;
    }

    return detachOutcome.IsSuccess();
```

- For API details, see DetachRolePolicy in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To detach a policy from a role

The following `detach-role-policy` example removes the managed policy with the ARN `arn:aws:iam::123456789012:policy/FederatedTesterAccessPolicy` from the role called `FedTesterRole`.

```
aws iam detach-role-policy \
    --role-name FedTesterRole \
    --policy-arn arn:aws:iam::123456789012:policy/FederatedTesterAccessPolicy
```

This command produces no output.

For more information, see IAM role management in the *AWS IAM User Guide*.

- For API details, see DetachRolePolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
```

```
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(ctx context.Context, roleName string,
 policyArn string) error {
 _, err := wrapper.IamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName,
 err)
 }
 return err
}
```

- For API details, see DetachRolePolicy in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.DetachRolePolicyRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
```

```
import software.amazon.awssdk.services.iam.model.IamException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetachRolePolicy {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <roleName> <policyArn>\s

                Where:
                    roleName - A role name that you can obtain from the AWS
 Management Console.\s
                    policyArn - A policy ARN that you can obtain from the AWS
 Management Console.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String roleName = args[0];
        String policyArn = args[1];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();
        detachPolicy(iam, roleName, policyArn);
        System.out.println("Done");
        iam.close();
    }

    public static void detachPolicy(IamClient iam, String roleName, String
 policyArn) {
        try {
```

```
            DetachRolePolicyRequest request = DetachRolePolicyRequest.builder()
                    .roleName(roleName)
                    .policyArn(policyArn)
                    .build();

            iam.detachRolePolicy(request);
            System.out.println("Successfully detached policy " + policyArn +
                    " from role " + roleName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DetachRolePolicy in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Detach the policy.

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
```

```
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DetachRolePolicy in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
          RoleName: process.argv[2],
        };
```

```
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DetachRolePolicy in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun detachPolicy(
    roleNameVal: String,
    policyArnVal: String,
) {
    val request =
        DetachRolePolicyRequest {
            roleName = roleNameVal
            policyArn = policyArnVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.detachRolePolicy(request)
        println("Successfully detached policy $policyArnVal from role
 $roleNameVal")
```

```
    }
}
```

- For API details, see DetachRolePolicy in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example detaches the managed group policy whose ARN is `arn:aws:iam::123456789012:policy/FederatedTesterAccessPolicy` from the role named `FedTesterRole`.

```
Unregister-IAMRolePolicy -RoleName FedTesterRole -PolicyArn
 arn:aws:iam::123456789012:policy/FederatedTesterAccessPolicy
```

Example 2: This example finds all of the managed policies that are attached to the role named `FedTesterRole` and detaches them from the role.

```
Get-IAMAttachedRolePolicyList -RoleName FedTesterRole | Unregister-IAMRolePolicy
 -Rolename FedTesterRole
```

- For API details, see DetachRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example detaches the managed group policy whose ARN is `arn:aws:iam::123456789012:policy/FederatedTesterAccessPolicy` from the role named `FedTesterRole`.

```
Unregister-IAMRolePolicy -RoleName FedTesterRole -PolicyArn
 arn:aws:iam::123456789012:policy/FederatedTesterAccessPolicy
```

Example 2: This example finds all of the managed policies that are attached to the role named `FedTesterRole` and detaches them from the role.

```
Get-IAMAttachedRolePolicyList -RoleName FedTesterRole | Unregister-IAMRolePolicy
 -Rolename FedTesterRole
```

- For API details, see DetachRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Detach a policy from a role using the Boto3 Policy object.

```
def detach_from_role(role_name, policy_arn):
    """
    Detaches a policy from a role.

    :param role_name: The name of the role. **Note** this is the name, not the
 ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Policy(policy_arn).detach_role(RoleName=role_name)
        logger.info("Detached policy %s from role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from role %s.", policy_arn, role_name
        )
        raise
```

Detach a policy from a role using the Boto3 Role object.

```
def detach_policy(role_name, policy_arn):
    """
    Detaches a policy from a role.

    :param role_name: The name of the role. **Note** this is the name, not the
 ARN.
    :param policy_arn: The ARN of the policy.
    """
```

```
    try:
        iam.Role(role_name).detach_policy(PolicyArn=policy_arn)
        logger.info("Detached policy %s from role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from role %s.", policy_arn, role_name
        )
        raise
```

- For API details, see DetachRolePolicy in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, attaches, and detaches role policies.

```
# Manages policies in AWS Identity and Access Management (IAM)
class RolePolicyManager
  # Initialize with an AWS IAM client
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'PolicyManager'
  end

  # Creates a policy
  #
  # @param policy_name [String] The name of the policy
  # @param policy_document [Hash] The policy document
  # @return [String] The policy ARN if successful, otherwise nil
```

```
  def create_policy(policy_name, policy_document)
    response = @iam_client.create_policy(
      policy_name: policy_name,
      policy_document: policy_document.to_json
    )
    response.policy.arn
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating policy: #{e.message}")
    nil
  end

  # Fetches an IAM policy by its ARN
  # @param policy_arn [String] the ARN of the IAM policy to retrieve
  # @return [Aws::IAM::Types::GetPolicyResponse] the policy object if found
  def get_policy(policy_arn)
    response = @iam_client.get_policy(policy_arn: policy_arn)
    policy = response.policy
    @logger.info("Got policy '#{policy.policy_name}'. Its ID is:
#{policy.policy_id}.")
    policy
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Couldn't get policy '#{policy_arn}'. The policy does not
exist.")
    raise
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't get policy '#{policy_arn}'. Here's why: #{e.code}:
#{e.message}")
    raise
  end

  # Attaches a policy to a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def attach_policy_to_role(role_name, policy_arn)
    @iam_client.attach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error attaching policy to role: #{e.message}")
    false
```

```
  end

  # Lists policy ARNs attached to a role
  #
  # @param role_name [String] The name of the role
  # @return [Array<String>] List of policy ARNs
  def list_attached_policy_arns(role_name)
    response = @iam_client.list_attached_role_policies(role_name: role_name)
    response.attached_policies.map(&:policy_arn)
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing policies attached to role: #{e.message}")
    []
  end

  # Detaches a policy from a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def detach_policy_from_role(role_name, policy_arn)
    @iam_client.detach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error detaching policy from role: #{e.message}")
    false
  end
end
```

- For API details, see DetachRolePolicy in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn detach_role_policy(
    client: &iamClient,
    role_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .detach_role_policy()
        .role_name(role_name)
        .policy_arn(policy_arn)
        .send()
        .await?;

    Ok(())
}
```

- For API details, see DetachRolePolicy in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->detachrolepolicy(
          iv_rolename = iv_role_name
          iv_policyarn = iv_policy_arn ).
        MESSAGE 'Policy detached from role successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Role or policy does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DetachRolePolicy in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func detachRolePolicy(policy: IAMClientTypes.Policy, role:
 IAMClientTypes.Role) async throws {
        let input = DetachRolePolicyInput(
            policyArn: policy.arn,
            roleName: role.roleName
        )

        do {
            _ = try await iamClient.detachRolePolicy(input: input)
        } catch {
            print("ERROR: detachRolePolicy:", dump(error))
            throw error
        }
    }
```

- For API details, see DetachRolePolicy in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DetachUserPolicy` with an AWS SDK or CLI

The following code examples show how to use `DetachUserPolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create read-only and read-write users

CLI

AWS CLI

To detach a policy from a user

The following `detach-user-policy` example removes the managed policy with the ARN `arn:aws:iam::123456789012:policy/TesterPolicy` from the user Bob.

```
aws iam detach-user-policy \
    --user-name Bob \
    --policy-arn arn:aws:iam::123456789012:policy/TesterPolicy
```

This command produces no output.

For more information, see Changing permissions for an IAM user in the *AWS IAM User Guide*.

- For API details, see DetachUserPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example detaches the managed policy whose ARN is `arn:aws:iam::123456789012:policy/TesterPolicy` from the IAM user named Bob.

```
Unregister-IAMUserPolicy -UserName Bob -PolicyArn
 arn:aws:iam::123456789012:policy/TesterPolicy
```

Example 2: This example finds all the managed policies that are attached to the IAM user named Theresa and detaches those policies from the user.

```
Get-IAMAttachedUserPolicyList -UserName Theresa | Unregister-IAMUserPolicy -
Username Theresa
```

- For API details, see DetachUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example detaches the managed policy whose ARN is `arn:aws:iam::123456789012:policy/TesterPolicy` from the IAM user named Bob.

```
Unregister-IAMUserPolicy -UserName Bob -PolicyArn
 arn:aws:iam::123456789012:policy/TesterPolicy
```

Example 2: This example finds all the managed policies that are attached to the IAM user named Theresa and detaches those policies from the user.

```
Get-IAMAttachedUserPolicyList -UserName Theresa | Unregister-IAMUserPolicy -
Username Theresa
```

- For API details, see DetachUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def detach_policy(user_name, policy_arn):
    """
    Detaches a policy from a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).detach_policy(PolicyArn=policy_arn)
        logger.info("Detached policy %s from user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from user %s.", policy_arn, user_name
        )
        raise
```

- For API details, see DetachUserPolicy in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
 # Detaches a policy from a user
 #
 # @param user_name [String] The name of the user
 # @param policy_arn [String] The ARN of the policy to detach
 # @return [Boolean] true if the policy was successfully detached, false
otherwise
 def detach_user_policy(user_name, policy_arn)
   @iam_client.detach_user_policy(
     user_name: user_name,
     policy_arn: policy_arn
   )
   @logger.info("Policy '#{policy_arn}' detached from user '#{user_name}'
successfully.")
   true
 rescue Aws::IAM::Errors::NoSuchEntity
   @logger.error('Error detaching policy: Policy or user does not exist.')
   false
 rescue Aws::IAM::Errors::ServiceError => e
   @logger.error("Error detaching policy from user '#{user_name}':
#{e.message}")
   false
 end
```

- For API details, see DetachUserPolicy in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn detach_user_policy(
    client: &iamClient,
    user_name: &str,
    policy_arn: &str,
) -> Result<(), iamError> {
    client
        .detach_user_policy()
        .user_name(user_name)
        .policy_arn(policy_arn)
        .send()
        .await?;

    Ok(())
}
```

- For API details, see DetachUserPolicy in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->detachuserpolicy(
```

```
          iv_username = iv_user_name
          iv_policyarn = iv_policy_arn ).
        MESSAGE 'Policy detached from user successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'User or policy does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see DetachUserPolicy in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `EnableMfaDevice` with a CLI

The following code examples show how to use `EnableMfaDevice`.

CLI

AWS CLI

To enable an MFA device

The following `enable-mfa-device` example assigns the MFA device with the serial number `arn:aws:iam::123456789012:mfa/BobsMFADevice` to the user Bob. The command also synchronizes the device with AWS by including the first two codes in sequence from the virtual MFA device. Before you assign a virtual MFA device to a user, you create it by using the `create-virtual-mfa-device` command.

```
aws iam enable-mfa-device \
    --user-name Bob \
    --serial-number arn:aws:iam::123456789012:mfa/BobsMFADevice \
    --authentication-code1 123456 \
    --authentication-code2 789012
```

This command produces no output.

For more information, see Enabling a virtual multi-factor authentication (MFA) device in the *AWS IAM User Guide*.

- For API details, see EnableMfaDevice in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command enables the hardware MFA device with the serial number 987654321098 and associates the device with the user Bob. It includes the first two codes in sequence from the device.

```
Enable-IAMMFADevice -UserName "Bob" -SerialNumber "987654321098" -
AuthenticationCode1 "12345678" -AuthenticationCode2 "87654321"
```

Example 2: This example creates and enables a virtual MFA device. The first command creates the virtual device and returns the device's object representation in the variable `$MFADevice`. You can use the `.Base32StringSeed` or `QRCodePng` properties to configure the user's software application. The final command assigns the device to the user `David`, identifying the device by its serial number. The command also synchronizes the device with AWS by including the first two codes in sequence from the virtual MFA device.

```
$MFADevice = New-IAMVirtualMFADevice -VirtualMFADeviceName "MyMFADevice"
# see example for New-IAMVirtualMFADevice to see how to configure the software
 program with PNG or base32 seed code
Enable-IAMMFADevice -UserName "David" -SerialNumber -SerialNumber
 $MFADevice.SerialNumber -AuthenticationCode1 "24681357" -AuthenticationCode2
 "13572468"
```

- For API details, see [EnableMfaDevice] in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command enables the hardware MFA device with the serial number 987654321098 and associates the device with the user Bob. It includes the first two codes in sequence from the device.

```
Enable-IAMMFADevice -UserName "Bob" -SerialNumber "987654321098" -
AuthenticationCode1 "12345678" -AuthenticationCode2 "87654321"
```

Example 2: This example creates and enables a virtual MFA device. The first command creates the virtual device and returns the device's object representation in the variable `$MFADevice`. You can use the `.Base32StringSeed` or `QRCodePng` properties to

configure the user's software application. The final command assigns the device to the user `David`, identifying the device by its serial number. The command also synchronizes the device with AWS by including the first two codes in sequence from the virtual MFA device.

```
$MFADevice = New-IAMVirtualMFADevice -VirtualMFADeviceName "MyMFADevice"
# see example for New-IAMVirtualMFADevice to see how to configure the software
 program with PNG or base32 seed code
Enable-IAMMFADevice -UserName "David" -SerialNumber -SerialNumber
 $MFADevice.SerialNumber -AuthenticationCode1 "24681357" -AuthenticationCode2
 "13572468"
```

- For API details, see EnableMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GenerateCredentialReport` with an AWS SDK or CLI

The following code examples show how to use `GenerateCredentialReport`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage your account

CLI

AWS CLI

To generate a credential report

The following `generate-credential-report` example attempts to generate a credential report for the AWS account.

```
aws iam generate-credential-report
```

Output:

```
{
    "State":  "STARTED",
    "Description": "No report exists. Starting a new report generation task"
}
```

For more information, see Getting credential reports for your AWS account in the *AWS IAM User Guide*.

- For API details, see GenerateCredentialReport in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example requests generation of a new report, which can be done every four hours. If the last report is still recent the State field reads `COMPLETE`. Use `Get-IAMCredentialReport` to view the completed report.

```
Request-IAMCredentialReport
```

Output:

```
Description                                                    State
-----------                                                    -----
No report exists. Starting a new report generation task        STARTED
```

- For API details, see GenerateCredentialReport in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example requests generation of a new report, which can be done every four hours. If the last report is still recent the State field reads `COMPLETE`. Use `Get-IAMCredentialReport` to view the completed report.

```
Request-IAMCredentialReport
```

Output:

```
Description                                                    State
```

```
-----------                                                    -----
No report exists. Starting a new report generation task        STARTED
```

- For API details, see GenerateCredentialReport in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def generate_credential_report():
    """
    Starts generation of a credentials report about the current account. After
    calling this function to generate the report, call get_credential_report
    to get the latest report. A new report can be generated a minimum of four
 hours
    after the last one was generated.
    """
    try:
        response = iam.meta.client.generate_credential_report()
        logger.info(
            "Generating credentials report for your account. " "Current state is
 %s.",
            response["State"],
        )
    except ClientError:
        logger.exception("Couldn't generate a credentials report for your
 account.")
        raise
    else:
        return response
```

- For API details, see GenerateCredentialReport in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->generatecredentialreport( ).
    MESSAGE 'Credential report generation started.' TYPE 'I'.
  CATCH /aws1/cx_iamlimitexceededex.
    MESSAGE 'Report generation limit exceeded.' TYPE 'E'.
  CATCH /aws1/cx_iamservicefailureex.
    MESSAGE 'Service failure when generating credential report.' TYPE 'E'.
ENDTRY.
```

- For API details, see GenerateCredentialReport in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GenerateServiceLastAccessedDetails` with a CLI

The following code examples show how to use `GenerateServiceLastAccessedDetails`.

CLI

AWS CLI

Example 1: To generate a service access report for a custom policy

The following `generate-service-last-accessed-details` example starts a background job to generate a report that lists the services accessed by IAM users and other entities with a custom policy named `intern-boundary`. You can display the report after it is created by running the `get-service-last-accessed-details` command.

```
aws iam generate-service-last-accessed-details \
    --arn arn:aws:iam::123456789012:policy/intern-boundary
```

Output:

```
{
    "JobId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Example 2: To generate a service access report for the AWS managed AdministratorAccess policy

The following `generate-service-last-accessed-details` example starts a background job to generate a report that lists the services accessed by IAM users and other entities with the AWS managed `AdministratorAccess` policy. You can display the report after it is created by running the `get-service-last-accessed-details` command.

```
aws iam generate-service-last-accessed-details \
    --arn arn:aws:iam::aws:policy/AdministratorAccess
```

Output:

```
{
    "JobId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

For more information, see Refining permissions in AWS using last accessed information in the *AWS IAM User Guide*.

- For API details, see GenerateServiceLastAccessedDetails in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example is equivalent cmdlet of GenerateServiceLastAccessedDetails API. This provides with a job id which can be used in Get-IAMServiceLastAccessedDetail and Get-IAMServiceLastAccessedDetailWithEntity

```
Request-IAMServiceLastAccessedDetail -Arn arn:aws:iam::123456789012:user/TestUser
```

- For API details, see GenerateServiceLastAccessedDetails in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example is equivalent cmdlet of GenerateServiceLastAccessedDetails API. This provides with a job id which can be used in Get-IAMServiceLastAccessedDetail and Get-IAMServiceLastAccessedDetailWithEntity

```
Request-IAMServiceLastAccessedDetail -Arn arn:aws:iam::123456789012:user/TestUser
```

- For API details, see GenerateServiceLastAccessedDetails in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetAccessKeyLastUsed` with an AWS SDK or CLI

The following code examples show how to use `GetAccessKeyLastUsed`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage access keys

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::accessKeyLastUsed(const Aws::String &secretKeyID,
                                    const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::GetAccessKeyLastUsedRequest request;

    request.SetAccessKeyId(secretKeyID);

    Aws::IAM::Model::GetAccessKeyLastUsedOutcome outcome =
 iam.GetAccessKeyLastUsed(
            request);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error querying last used time for access key " <<
                  secretKeyID << ":" << outcome.GetError().GetMessage() <<
 std::endl;
    }
    else {
        Aws::String lastUsedTimeString =
                outcome.GetResult()
                        .GetAccessKeyLastUsed()
                        .GetLastUsedDate()
                        .ToGmtString(Aws::Utils::DateFormat::ISO_8601);
        std::cout << "Access key " << secretKeyID << " last used at time " <<
                  lastUsedTimeString << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see GetAccessKeyLastUsed in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To retrieve information about when the specified access key was last used

The following `get-access-key-last-used` example retrieves information about when the access key `AKIAIOSFODNN7EXAMPLE` was last used.

```
aws iam get-access-key-last-used \
    --access-key-id AKIAIOSFODNN7EXAMPLE
```

Output:

```
{
    "UserName":  "Bob",
    "AccessKeyLastUsed": {
        "Region": "us-west-2",
        "ServiceName": "iam",
        "LastUsedDate": "2015-06-16T22:45:00Z"
    }
}
```

For more information, see Managing access keys for IAM users in the *AWS IAM User Guide*.

- For API details, see GetAccessKeyLastUsed in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get the access key.

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });

  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }

  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetAccessKeyLastUsed in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });
```

```
iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetAccessKeyLastUsed in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns the owning user name and last-usage information for the supplied access key.

```
Get-IAMAccessKeyLastUsed -AccessKeyId ABCDEXAMPLE
```

- For API details, see GetAccessKeyLastUsed in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns the owning user name and last-usage information for the supplied access key.

```
Get-IAMAccessKeyLastUsed -AccessKeyId ABCDEXAMPLE
```

- For API details, see GetAccessKeyLastUsed in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_last_use(key_id):
    """
    Gets information about when and how a key was last used.

    :param key_id: The ID of the key to look up.
    :return: Information about the key's last use.
    """
    try:
        response = iam.meta.client.get_access_key_last_used(AccessKeyId=key_id)
        last_used_date = response["AccessKeyLastUsed"].get("LastUsedDate", None)
        last_service = response["AccessKeyLastUsed"].get("ServiceName", None)
        logger.info(
            "Key %s was last used by %s on %s to access %s.",
            key_id,
            response["UserName"],
            last_used_date,
            last_service,
        )
    except ClientError:
        logger.exception("Couldn't get last use of key %s.", key_id)
        raise
    else:
        return response
```

- For API details, see GetAccessKeyLastUsed in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->getaccesskeylastused(
      iv_accesskeyid = iv_access_key_id ).
    MESSAGE 'Retrieved access key last used information.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'Access key does not exist.' TYPE 'E'.
ENDTRY.
```

- For API details, see GetAccessKeyLastUsed in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetAccountAuthorizationDetails` with an AWS SDK or CLI

The following code examples show how to use `GetAccountAuthorizationDetails`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage your account

CLI

AWS CLI

To list an AWS account's IAM users, groups, roles, and policies

The following `get-account-authorization-details` example returns information about all IAM users, groups, roles, and policies in the AWS account.

```
aws iam get-account-authorization-details
```

Output:

```
{
    "RoleDetailList": [
        {
            "AssumeRolePolicyDocument": {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Sid": "",
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "ec2.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "RoleId": "AROA1234567890EXAMPLE",
            "CreateDate": "2014-07-30T17:09:20Z",
            "InstanceProfileList": [
                {
                    "InstanceProfileId": "AIPA1234567890EXAMPLE",
                    "Roles": [
                        {
                            "AssumeRolePolicyDocument": {
                                "Version":"2012-10-17",
                                "Statement": [
                                    {
                                        "Sid": "",
                                        "Effect": "Allow",
                                        "Principal": {
                                            "Service": "ec2.amazonaws.com"
                                        },
                                        "Action": "sts:AssumeRole"
                                    }
                                ]
                            },
```

```
                            "RoleId": "AROA1234567890EXAMPLE",
                            "CreateDate": "2014-07-30T17:09:20Z",
                            "RoleName": "EC2role",
                            "Path": "/",
                            "Arn": "arn:aws:iam::123456789012:role/EC2role"
                        }
                    ],
                    "CreateDate": "2014-07-30T17:09:20Z",
                    "InstanceProfileName": "EC2role",
                    "Path": "/",
                    "Arn": "arn:aws:iam::123456789012:instance-profile/EC2role"
                }
            ],
            "RoleName": "EC2role",
            "Path": "/",
            "AttachedManagedPolicies": [
                {
                    "PolicyName": "AmazonS3FullAccess",
                    "PolicyArn": "arn:aws:iam::aws:policy/AmazonS3FullAccess"
                },
                {
                    "PolicyName": "AmazonDynamoDBFullAccess",
                    "PolicyArn": "arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess"
                }
            ],
            "RoleLastUsed": {
                "Region": "us-west-2",
                "LastUsedDate": "2019-11-13T17:30:00Z"
            },
            "RolePolicyList": [],
            "Arn": "arn:aws:iam::123456789012:role/EC2role"
        }
    ],
    "GroupDetailList": [
        {
            "GroupId": "AIDA1234567890EXAMPLE",
            "AttachedManagedPolicies": [
                {
                    "PolicyName": "AdministratorAccess",
                    "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
                }
            ],
            "GroupName": "Admins",
```

```
            "Path": "/",
            "Arn": "arn:aws:iam::123456789012:group/Admins",
            "CreateDate": "2013-10-14T18:32:24Z",
            "GroupPolicyList": []
        },
        {
            "GroupId": "AIDA1234567890EXAMPLE",
            "AttachedManagedPolicies": [
                {
                    "PolicyName": "PowerUserAccess",
                    "PolicyArn": "arn:aws:iam::aws:policy/PowerUserAccess"
                }
            ],
            "GroupName": "Dev",
            "Path": "/",
            "Arn": "arn:aws:iam::123456789012:group/Dev",
            "CreateDate": "2013-10-14T18:33:55Z",
            "GroupPolicyList": []
        },
        {
            "GroupId": "AIDA1234567890EXAMPLE",
            "AttachedManagedPolicies": [],
            "GroupName": "Finance",
            "Path": "/",
            "Arn": "arn:aws:iam::123456789012:group/Finance",
            "CreateDate": "2013-10-14T18:57:48Z",
            "GroupPolicyList": [
                {
                    "PolicyName": "policygen-20131014",
                    "PolicyDocument": {
                        "Version":"2012-10-17",
                        "Statement": [
                            {
                                "Action": "aws-portal:*",
                                "Sid": "Stmt1381777017000",
                                "Resource": "*",
                                "Effect": "Allow"
                            }
                        ]
                    }
                }
            ]
        }
    ],
```

```
    "UserDetailList": [
        {
            "UserName": "Alice",
            "GroupList": [
                "Admins"
            ],
            "CreateDate": "2013-10-14T18:32:24Z",
            "UserId": "AIDA1234567890EXAMPLE",
            "UserPolicyList": [],
            "Path": "/",
            "AttachedManagedPolicies": [],
            "Arn": "arn:aws:iam::123456789012:user/Alice"
        },
        {
            "UserName": "Bob",
            "GroupList": [
                "Admins"
            ],
            "CreateDate": "2013-10-14T18:32:25Z",
            "UserId": "AIDA1234567890EXAMPLE",
            "UserPolicyList": [
                {
                    "PolicyName": "DenyBillingAndIAMPolicy",
                    "PolicyDocument": {
                        "Version":"2012-10-17",
                        "Statement": {
                            "Effect": "Deny",
                            "Action": [
                                "aws-portal:*",
                                "iam:*"
                            ],
                            "Resource": "*"
                        }
                    }
                }
            ],
            "Path": "/",
            "AttachedManagedPolicies": [],
            "Arn": "arn:aws:iam::123456789012:user/Bob"
        },
        {
            "UserName": "Charlie",
            "GroupList": [
                "Dev"
```

```
            ],
            "CreateDate": "2013-10-14T18:33:56Z",
            "UserId": "AIDA1234567890EXAMPLE",
            "UserPolicyList": [],
            "Path": "/",
            "AttachedManagedPolicies": [],
            "Arn": "arn:aws:iam::123456789012:user/Charlie"
        }
    ],
    "Policies": [
        {
            "PolicyName": "create-update-delete-set-managed-policies",
            "CreateDate": "2015-02-06T19:58:34Z",
            "AttachmentCount": 1,
            "IsAttachable": true,
            "PolicyId": "ANPA1234567890EXAMPLE",
            "DefaultVersionId": "v1",
            "PolicyVersionList": [
                {
                    "CreateDate": "2015-02-06T19:58:34Z",
                    "VersionId": "v1",
                    "Document": {
                        "Version":"2012-10-17",
                        "Statement": {
                            "Effect": "Allow",
                            "Action": [
                                "iam:CreatePolicy",
                                "iam:CreatePolicyVersion",
                                "iam:DeletePolicy",
                                "iam:DeletePolicyVersion",
                                "iam:GetPolicy",
                                "iam:GetPolicyVersion",
                                "iam:ListPolicies",
                                "iam:ListPolicyVersions",
                                "iam:SetDefaultPolicyVersion"
                            ],
                            "Resource": "*"
                        }
                    },
                    "IsDefaultVersion": true
                }
            ],
            "Path": "/",
```

```
            "Arn": "arn:aws:iam::123456789012:policy/create-update-delete-set-
managed-policies",
            "UpdateDate": "2015-02-06T19:58:34Z"
        },
        {
            "PolicyName": "S3-read-only-specific-bucket",
            "CreateDate": "2015-01-21T21:39:41Z",
            "AttachmentCount": 1,
            "IsAttachable": true,
            "PolicyId": "ANPA1234567890EXAMPLE",
            "DefaultVersionId": "v1",
            "PolicyVersionList": [
                {
                    "CreateDate": "2015-01-21T21:39:41Z",
                    "VersionId": "v1",
                    "Document": {
                        "Version":"2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "s3:Get*",
                                    "s3:List*"
                                ],
                                "Resource": [
                                    "arn:aws:s3:::amzn-s3-demo-bucket",
                                    "arn:aws:s3:::amzn-s3-demo-bucket/*"
                                ]
                            }
                        ]
                    },
                    "IsDefaultVersion": true
                }
            ],
            "Path": "/",
            "Arn": "arn:aws:iam::123456789012:policy/S3-read-only-specific-
bucket",
            "UpdateDate": "2015-01-21T23:39:41Z"
        },
        {
            "PolicyName": "AmazonEC2FullAccess",
            "CreateDate": "2015-02-06T18:40:15Z",
            "AttachmentCount": 1,
            "IsAttachable": true,
```

```
            "PolicyId": "ANPA1234567890EXAMPLE",
            "DefaultVersionId": "v1",
            "PolicyVersionList": [
                {
                    "CreateDate": "2014-10-30T20:59:46Z",
                    "VersionId": "v1",
                    "Document": {
                        "Version":"2012-10-17",
                        "Statement": [
                            {
                                "Action": "ec2:*",
                                "Effect": "Allow",
                                "Resource": "*"
                            },
                            {
                                "Effect": "Allow",
                                "Action": "elasticloadbalancing:*",
                                "Resource": "*"
                            },
                            {
                                "Effect": "Allow",
                                "Action": "cloudwatch:*",
                                "Resource": "*"
                            },
                            {
                                "Effect": "Allow",
                                "Action": "autoscaling:*",
                                "Resource": "*"
                            }
                        ]
                    },
                    "IsDefaultVersion": true
                }
            ],
            "Path": "/",
            "Arn": "arn:aws:iam::aws:policy/AmazonEC2FullAccess",
            "UpdateDate": "2015-02-06T18:40:15Z"
        }
    ],
    "Marker": "EXAMPLEkakv9BCuUNFDtxWSyfzetYwEx2ADc8dnzfvERF5S6YMvXKx41t6gCl/
eeaCX3Jo94/bKqezEAg8TEVS99EKFLxm3jtbpl25FDWEXAMPLE",
    "IsTruncated": true
}
```

For more information, see AWS security audit guidelines in the *AWS IAM User Guide*.

- For API details, see GetAccountAuthorizationDetails in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example gets authorization details about the identities in the AWS account, and displays the element list of the returned object, including users, groups, and roles. For example, the `UserDetailList` property displays details about the users. Similar information is available in the `RoleDetailList` and `GroupDetailList` properties.

```
$Details=Get-IAMAccountAuthorizationDetail
$Details
```

Output:

```
GroupDetailList : {Administrators, Developers, Testers, Backup}
IsTruncated     : False
Marker          :
RoleDetailList  : {TestRole1, AdminRole, TesterRole, clirole...}
UserDetailList  : {Administrator, Bob, BackupToS3, }
```

```
$Details.UserDetailList
```

Output:

```
Arn             : arn:aws:iam::123456789012:user/Administrator
CreateDate      : 10/16/2014 9:03:09 AM
GroupList       : {Administrators}
Path            : /
UserId          : AIDACKCEVSQ6CEXAMPLE1
UserName        : Administrator
UserPolicyList : {}

Arn             : arn:aws:iam::123456789012:user/Bob
CreateDate      : 4/6/2015 12:54:42 PM
GroupList       : {Developers}
Path            : /
```

```
UserId          : AIDACKCEVSQ6CEXAMPLE2
UserName        : bab
UserPolicyList : {}

Arn             : arn:aws:iam::123456789012:user/BackupToS3
CreateDate      : 1/27/2015 10:15:08 AM
GroupList       : {Backup}
Path            : /
UserId          : AIDACKCEVSQ6CEXAMPLE3
UserName        : BackupToS3
UserPolicyList : {BackupServicePermissionsToS3Buckets}
```

- For API details, see GetAccountAuthorizationDetails in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example gets authorization details about the identities in the AWS account, and displays the element list of the returned object, including users, groups, and roles. For example, the `UserDetailList` property displays details about the users. Similar information is available in the `RoleDetailList` and `GroupDetailList` properties.

```
$Details=Get-IAMAccountAuthorizationDetail
$Details
```

Output:

```
GroupDetailList : {Administrators, Developers, Testers, Backup}
IsTruncated     : False
Marker          :
RoleDetailList  : {TestRole1, AdminRole, TesterRole, clirole...}
UserDetailList  : {Administrator, Bob, BackupToS3, }
```

```
$Details.UserDetailList
```

Output:

```
Arn             : arn:aws:iam::123456789012:user/Administrator
CreateDate      : 10/16/2014 9:03:09 AM
GroupList       : {Administrators}
```

```
Path           : /
UserId         : AIDACKCEVSQ6CEXAMPLE1
UserName       : Administrator
UserPolicyList : {}

Arn            : arn:aws:iam::123456789012:user/Bob
CreateDate     : 4/6/2015 12:54:42 PM
GroupList      : {Developers}
Path           : /
UserId         : AIDACKCEVSQ6CEXAMPLE2
UserName       : bab
UserPolicyList : {}

Arn            : arn:aws:iam::123456789012:user/BackupToS3
CreateDate     : 1/27/2015 10:15:08 AM
GroupList      : {Backup}
Path           : /
UserId         : AIDACKCEVSQ6CEXAMPLE3
UserName       : BackupToS3
UserPolicyList : {BackupServicePermissionsToS3Buckets}
```

- For API details, see GetAccountAuthorizationDetails in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_authorization_details(response_filter):
    """
    Gets an authorization detail report for the current account.

    :param response_filter: A list of resource types to include in the report,
 such
                            as users or roles. When not specified, all resources
```

```
                            are included.
    :return: The authorization detail report.
    """
    try:
        account_details = iam.meta.client.get_account_authorization_details(
            Filter=response_filter
        )
        logger.debug(account_details)
    except ClientError:
        logger.exception("Couldn't get details for your account.")
        raise
    else:
        return account_details
```

- For API details, see GetAccountAuthorizationDetails in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->getaccountauthdetails( ).
        MESSAGE 'Retrieved account authorization details.' TYPE 'I'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when getting account authorization details.'
TYPE 'E'.
    ENDTRY.
```

- For API details, see GetAccountAuthorizationDetails in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetAccountPasswordPolicy` with an AWS SDK or CLI

The following code examples show how to use `GetAccountPasswordPolicy`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To see the current account password policy

The following `get-account-password-policy` example displays details about the password policy for the current account.

```
aws iam get-account-password-policy
```

Output:

```
{
    "PasswordPolicy": {
        "AllowUsersToChangePassword": false,
        "RequireLowercaseCharacters": false,
        "RequireUppercaseCharacters": false,
        "MinimumPasswordLength": 8,
        "RequireNumbers": true,
        "RequireSymbols": true
    }
}
```

If no password policy is defined for the account, the command returns a `NoSuchEntity` error.

For more information, see Setting an account password policy for IAM users in the *AWS IAM User Guide*.

- For API details, see GetAccountPasswordPolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)
```

```
// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// GetAccountPasswordPolicy gets the account password policy for the current
 account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy(ctx context.Context)
 (*types.PasswordPolicy, error) {
 var pwPolicy *types.PasswordPolicy
 result, err := wrapper.IamClient.GetAccountPasswordPolicy(ctx,
  &iam.GetAccountPasswordPolicyInput{})
 if err != nil {
  log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
 } else {
  pwPolicy = result.PasswordPolicy
 }
 return pwPolicy, err
}
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get the account password policy.

```
import {
  GetAccountPasswordPolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
  const command = new GetAccountPasswordPolicyCommand({});

  const response = await client.send(command);
  console.log(response.PasswordPolicy);
  return response;
};
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function getAccountPasswordPolicy()
    {
        return $this->iamClient->getAccountPasswordPolicy();
    }
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details about the password policy for the current account. If no password policy is defined for the account, the command returns a `NoSuchEntity` error.

```
Get-IAMAccountPasswordPolicy
```

Output:

```
AllowUsersToChangePassword : True
ExpirePasswords            : True
HardExpiry                 : False
MaxPasswordAge             : 90
MinimumPasswordLength      : 8
PasswordReusePrevention    : 20
RequireLowercaseCharacters : True
RequireNumbers             : True
RequireSymbols             : False
RequireUppercaseCharacters : True
```

- For API details, see GetAccountPasswordPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details about the password policy for the current account. If no password policy is defined for the account, the command returns a `NoSuchEntity` error.

```
Get-IAMAccountPasswordPolicy
```

Output:

```
AllowUsersToChangePassword : True
ExpirePasswords            : True
HardExpiry                 : False
MaxPasswordAge             : 90
MinimumPasswordLength      : 8
PasswordReusePrevention    : 20
```

```
RequireLowercaseCharacters : True
RequireNumbers             : True
RequireSymbols             : False
RequireUppercaseCharacters : True
```

- For API details, see GetAccountPasswordPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def print_password_policy():
    """
    Prints the password policy for the account.
    """
    try:
        pw_policy = iam.AccountPasswordPolicy()
        print("Current account password policy:")
        print(
            f"\tallow_users_to_change_password:
 {pw_policy.allow_users_to_change_password}"
        )
        print(f"\texpire_passwords: {pw_policy.expire_passwords}")
        print(f"\thard_expiry: {pw_policy.hard_expiry}")
        print(f"\tmax_password_age: {pw_policy.max_password_age}")
        print(f"\tminimum_password_length: {pw_policy.minimum_password_length}")
        print(f"\tpassword_reuse_prevention:
 {pw_policy.password_reuse_prevention}")
        print(
            f"\trequire_lowercase_characters:
 {pw_policy.require_lowercase_characters}"
        )
        print(f"\trequire_numbers: {pw_policy.require_numbers}")
        print(f"\trequire_symbols: {pw_policy.require_symbols}")
```

```
        print(
            f"\trequire_uppercase_characters:
 {pw_policy.require_uppercase_characters}"
        )
        printed = True
    except ClientError as error:
        if error.response["Error"]["Code"] == "NoSuchEntity":
            print("The account does not have a password policy set.")
        else:
            logger.exception("Couldn't get account password policy.")
            raise
    else:
        return printed
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Class to manage IAM account password policies
class PasswordPolicyManager
  attr_accessor :iam_client, :logger

  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'IAMPolicyManager'
  end

  # Retrieves and logs the account password policy
```

```
  def print_account_password_policy
    response = @iam_client.get_account_password_policy
    @logger.info("The account password policy is:
 #{response.password_policy.to_h}")
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.info('The account does not have a password policy.')
  rescue Aws::Errors::ServiceError => e
    @logger.error("Couldn't print the account password policy. Error: #{e.code} -
 #{e.message}")
    raise
  end
end
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn get_account_password_policy(
    client: &iamClient,
) -> Result<GetAccountPasswordPolicyOutput,
 SdkError<GetAccountPasswordPolicyError>> {
    let response = client.get_account_password_policy().send().await?;

    Ok(response)
}
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->getaccountpasswordpolicy( ).
    MESSAGE 'Retrieved account password policy.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'No password policy exists.' TYPE 'I'.
  CATCH /aws1/cx_iamservicefailureex.
    MESSAGE 'Service failure when getting password policy.' TYPE 'E'.
ENDTRY.
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetAccountSummary` with an AWS SDK or CLI

The following code examples show how to use `GetAccountSummary`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage your account

CLI

AWS CLI

To get information about IAM entity usage and IAM quotas in the current account

The following `get-account-summary` example returns information about the current IAM entity usage and current IAM entity quotas in the account.

```
aws iam get-account-summary
```

Output:

```
{
    "SummaryMap": {
        "UsersQuota": 5000,
        "GroupsQuota": 100,
        "InstanceProfiles": 6,
        "SigningCertificatesPerUserQuota": 2,
        "AccountAccessKeysPresent": 0,
        "RolesQuota": 250,
        "RolePolicySizeQuota": 10240,
        "AccountSigningCertificatesPresent": 0,
        "Users": 27,
        "ServerCertificatesQuota": 20,
        "ServerCertificates": 0,
        "AssumeRolePolicySizeQuota": 2048,
        "Groups": 7,
        "MFADevicesInUse": 1,
        "Roles": 3,
        "AccountMFAEnabled": 1,
        "MFADevices": 3,
        "GroupsPerUserQuota": 10,
        "GroupPolicySizeQuota": 5120,
        "InstanceProfilesQuota": 100,
        "AccessKeysPerUserQuota": 2,
        "Providers": 0,
        "UserPolicySizeQuota": 2048
    }
}
```

For more information about entity limitations, see IAM and AWS STS quotas in the *AWS IAM User Guide*.

- For API details, see GetAccountSummary in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns information about the current IAM entity usage and current IAM entity quotas in the AWS account.

```
Get-IAMAccountSummary
```

Output:

```
Key                                     Value
Users                                   7
GroupPolicySizeQuota                    5120
PolicyVersionsInUseQuota                10000
ServerCertificatesQuota                 20
AccountSigningCertificatesPresent       0
AccountAccessKeysPresent                0
Groups                                  3
UsersQuota                              5000
RolePolicySizeQuota                     10240
UserPolicySizeQuota                     2048
GroupsPerUserQuota                      10
AssumeRolePolicySizeQuota               2048
AttachedPoliciesPerGroupQuota           2
Roles                                   9
VersionsPerPolicyQuota                  5
GroupsQuota                             100
PolicySizeQuota                         5120
Policies                                5
RolesQuota                              250
ServerCertificates                      0
AttachedPoliciesPerRoleQuota            2
MFADevicesInUse                         2
PoliciesQuota                           1000
AccountMFAEnabled                       1
Providers                               2
InstanceProfilesQuota                   100
MFADevices                              4
AccessKeysPerUserQuota                  2
AttachedPoliciesPerUserQuota            2
SigningCertificatesPerUserQuota         2
```

```
PolicyVersionsInUse                     4
InstanceProfiles                        1
...
```

- For API details, see GetAccountSummary in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns information about the current IAM entity usage and current IAM entity quotas in the AWS account.

```
Get-IAMAccountSummary
```

Output:

```
Key                                     Value
Users                                   7
GroupPolicySizeQuota                    5120
PolicyVersionsInUseQuota                10000
ServerCertificatesQuota                 20
AccountSigningCertificatesPresent       0
AccountAccessKeysPresent                0
Groups                                  3
UsersQuota                              5000
RolePolicySizeQuota                     10240
UserPolicySizeQuota                     2048
GroupsPerUserQuota                      10
AssumeRolePolicySizeQuota               2048
AttachedPoliciesPerGroupQuota           2
Roles                                   9
VersionsPerPolicyQuota                  5
GroupsQuota                             100
PolicySizeQuota                         5120
Policies                                5
RolesQuota                              250
ServerCertificates                      0
AttachedPoliciesPerRoleQuota            2
MFADevicesInUse                         2
PoliciesQuota                           1000
AccountMFAEnabled                       1
Providers                               2
```

```
InstanceProfilesQuota                    100
MFADevices                               4
AccessKeysPerUserQuota                   2
AttachedPoliciesPerUserQuota             2
SigningCertificatesPerUserQuota          2
PolicyVersionsInUse                      4
InstanceProfiles                         1
...
```

- For API details, see GetAccountSummary in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_summary():
    """
    Gets a summary of account usage.

    :return: The summary of account usage.
    """
    try:
        summary = iam.AccountSummary()
        logger.debug(summary.summary_map)
    except ClientError:
        logger.exception("Couldn't get a summary for your account.")
        raise
    else:
        return summary.summary_map
```

- For API details, see GetAccountSummary in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->getaccountsummary( ).
    MESSAGE 'Retrieved account summary.' TYPE 'I'.
  CATCH /aws1/cx_iamservicefailureex.
    MESSAGE 'Service failure when getting account summary.' TYPE 'E'.
ENDTRY.
```

- For API details, see GetAccountSummary in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetContextKeysForCustomPolicy` with a CLI

The following code examples show how to use `GetContextKeysForCustomPolicy`.

CLI

AWS CLI

Example 1: To list the context keys referenced by one or more custom JSON policies provided as a parameter on the command line

The following `get-context-keys-for-custom-policy` example parses each supplied policy and lists the context keys used by those policies. Use this command to identify which context key values you must supply to successfully use the policy simulator commands `simulate-custom-policy` and `simulate-custom-policy`. You can also retrieve the list of context keys used by all policies associated by an IAM user or role by using the

`get-context-keys-for-custom-policy` command. Parameter values that begin with `file://` instruct the command to read the file and use the contents as the value for the parameter instead of the file name itself.

```
aws iam get-context-keys-for-custom-policy \
    --policy-input-list '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/${aws:username}","Condition":{"DateGreaterThan":
{"aws:CurrentTime":"2015-08-16T12:00:00Z"}}}}'
```

Output:

```
{
    "ContextKeyNames": [
        "aws:username",
        "aws:CurrentTime"
    ]
}
```

Example 2: To list the context keys referenced by one or more custom JSON policies provided as a file input

The following `get-context-keys-for-custom-policy` example lists the context keys in the same way as the previous example, except that the policies are provided in a file instead of as a parameter. Because the command expects a JSON list of strings, and not a list of JSON structures, the file must contain a JSON array in which each policy is a string. You must escape each embedded double-quote inside the policy string by preceding it with a backslash.

```
aws iam get-context-keys-for-custom-policy \
    --policy-input-list file://policyfile.json
```

Contents of `policyfile.json`:

```
[
    "{\"Version\": \"2012-10-17\", \"Statement\": {\"Effect\": \"Allow
\", \"Action\": \"dynamodb:*\", \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/${aws:username}\", \"Condition\": {\"DateGreaterThan\":
 {\"aws:CurrentTime\": \"2015-08-16T12:00:00Z\"}}}}"
]
```

Output:

```
{
    "ContextKeyNames": [
        "aws:username",
        "aws:CurrentTime"
    ]
}
```

For more information, see Using the IAM Policy Simulator (AWS CLI and AWS API) in the *AWS IAM User Guide*.

- For API details, see GetContextKeysForCustomPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example fetches all the context keys present in the provided policy json.In order to provide multiple policies you can provide as comma separated list of values.

```
$policy1 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/","Condition":{"DateGreaterThan":
{"aws:CurrentTime":"2015-08-16T12:00:00Z"}}}}'
$policy2 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/"}}'
Get-IAMContextKeysForCustomPolicy -PolicyInputList $policy1,$policy2
```

- For API details, see GetContextKeysForCustomPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example fetches all the context keys present in the provided policy json.In order to provide multiple policies you can provide as comma separated list of values.

```
$policy1 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
```

```
west-2:123456789012:table/","Condition":{"DateGreaterThan":
{"aws:CurrentTime":"2015-08-16T12:00:00Z"}}}}'
$policy2 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/"}}'
Get-IAMContextKeysForCustomPolicy -PolicyInputList $policy1,$policy2
```

- For API details, see GetContextKeysForCustomPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetContextKeysForPrincipalPolicy` with a CLI

The following code examples show how to use `GetContextKeysForPrincipalPolicy`.

CLI

AWS CLI

To list the context keys referenced by all policies associated with an IAM principal

The following `get-context-keys-for-principal-policy` example retrieves all policies that are attached to the user `saanvi` and any groups she is a member of. It then parses each and lists the context keys used by those policies. Use this command to identify which context key values you must supply to successfully use the `simulate-custom-policy` and `simulate-principal-policy` commands. You can also retrieve the list of context keys used by an arbitrary JSON policy by using the `get-context-keys-for-custom-policy` command.

```
aws iam get-context-keys-for-principal-policy \
    --policy-source-arn arn:aws:iam::123456789012:user/saanvi
```

Output:

```
{
    "ContextKeyNames": [
        "aws:username",
        "aws:CurrentTime"
```

```
    ]
}
```

For more information, see Using the IAM Policy Simulator (AWS CLI and AWS API) in the *AWS IAM User Guide*.

- For API details, see GetContextKeysForPrincipalPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example fetches all the context keys present in the provided policy json and the policies attached to IAM entity(user/role etc.). For -PolicyInputList you can provide multiple values list as comma separated values.

```
$policy1 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/","Condition":{"DateGreaterThan":
{"aws:CurrentTime":"2015-08-16T12:00:00Z"}}}}'
$policy2 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/"}}'
Get-IAMContextKeysForPrincipalPolicy -PolicyInputList $policy1,$policy2 -
PolicySourceArn arn:aws:iam::852640994763:user/TestUser
```

- For API details, see GetContextKeysForPrincipalPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example fetches all the context keys present in the provided policy json and the policies attached to IAM entity(user/role etc.). For -PolicyInputList you can provide multiple values list as comma separated values.

```
$policy1 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/","Condition":{"DateGreaterThan":
{"aws:CurrentTime":"2015-08-16T12:00:00Z"}}}}'
$policy2 = '{"Version":"2012-10-17",       "Statement":
{"Effect":"Allow","Action":"dynamodb:*","Resource":"arn:aws:dynamodb:us-
west-2:123456789012:table/"}}'
```

```
Get-IAMContextKeysForPrincipalPolicy -PolicyInputList $policy1,$policy2 -
PolicySourceArn arn:aws:iam::852640994763:user/TestUser
```

- For API details, see GetContextKeysForPrincipalPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetCredentialReport` with an AWS SDK or CLI

The following code examples show how to use `GetCredentialReport`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage your account

CLI

AWS CLI

To get a credential report

The following `get-credential-report` example opens the returned report and outputs it to the pipeline as an array of text lines.

```
aws iam get-credential-report
```

Output:

```
{
    "GeneratedTime":  "2015-06-17T19:11:50Z",
    "ReportFormat": "text/csv"
}
```

For more information, see Getting credential reports for your AWS account in the *AWS IAM User Guide*.

- For API details, see GetCredentialReport in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example opens the returned report and outputs it to the pipeline as an array of text lines. The first line is the header with comma-separated column names. Each successive row is the detail row for one user, with each field separated by commas. Before you can view the report, you must generate it with the `Request-IAMCredentialReport` cmdlet. To retrieve the report as a single string, use `-Raw` instead of `-AsTextArray`. The alias `-SplitLines` is also accepted for the `-AsTextArray` switch. For the full list of columns in the output consult the service API reference. Note that if you do not use `-AsTextArray` or `-SplitLines`, then you must extract the text from the `.Content` property using the .NET `StreamReader` class.

```
Request-IAMCredentialReport
```

Output:

```
Description                                                          State
-----------                                                          -----
No report exists. Starting a new report generation task              STARTED
```

```
Get-IAMCredentialReport -AsTextArray
```

Output:

```
 user,arn,user_creation_time,password_enabled,password_last_used,password_last_changed,pa
 root_account,arn:aws:iam::123456789012:root,2014-10-15T16:31:25+00:00,not_supported,2015
A,false,N/A,false,N/A,false,N/A
Administrator,arn:aws:iam::123456789012:user/
Administrator,2014-10-16T16:03:09+00:00,true,2015-04-20T15:18:32+00:00,2014-10-16T16:06:0
A,false,true,2014-12-03T18:53:41+00:00,true,2015-03-25T20:38:14+00:00,false,N/
A,false,N/A
Bill,arn:aws:iam::123456789012:user/Bill,2015-04-15T18:27:44+00:00,false,N/A,N/
A,N/A,false,false,N/A,false,N/A,false,2015-04-20T20:00:12+00:00,false,N/A
```

- For API details, see GetCredentialReport in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example opens the returned report and outputs it to the pipeline as an array of text lines. The first line is the header with comma-separated column names. Each successive row is the detail row for one user, with each field separated by commas. Before you can view the report, you must generate it with the `Request-IAMCredentialReport` cmdlet. To retrieve the report as a single string, use `-Raw` instead of `-AsTextArray`. The alias `-SplitLines` is also accepted for the `-AsTextArray` switch. For the full list of columns in the output consult the service API reference. Note that if you do not use `-AsTextArray` or `-SplitLines`, then you must extract the text from the `.Content` property using the .NET `StreamReader` class.

```
Request-IAMCredentialReport
```

Output:

```
Description                                                          State
-----------                                                          -----
No report exists. Starting a new report generation task              STARTED
```

```
Get-IAMCredentialReport -AsTextArray
```

Output:

```
 user,arn,user_creation_time,password_enabled,password_last_used,password_last_changed,pa
 root_account,arn:aws:iam::123456789012:root,2014-10-15T16:31:25+00:00,not_supported,2015
A,false,N/A,false,N/A,false,N/A
Administrator,arn:aws:iam::123456789012:user/
Administrator,2014-10-16T16:03:09+00:00,true,2015-04-20T15:18:32+00:00,2014-10-16T16:06:0
A,false,true,2014-12-03T18:53:41+00:00,true,2015-03-25T20:38:14+00:00,false,N/
A,false,N/A
Bill,arn:aws:iam::123456789012:user/Bill,2015-04-15T18:27:44+00:00,false,N/A,N/
A,N/A,false,false,N/A,false,N/A,false,2015-04-20T20:00:12+00:00,false,N/A
```

- For API details, see GetCredentialReport in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_credential_report():
    """
    Gets the most recently generated credentials report about the current
 account.

    :return: The credentials report.
    """
    try:
        response = iam.meta.client.get_credential_report()
        logger.debug(response["Content"])
    except ClientError:
        logger.exception("Couldn't get credentials report.")
        raise
    else:
        return response["Content"]
```

- For API details, see GetCredentialReport in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->getcredentialreport( ).
        MESSAGE 'Retrieved credential report.' TYPE 'I'.
      CATCH /aws1/cx_iamcredrptnotpresen00.
        MESSAGE 'Credential report not present.' TYPE 'E'.
      CATCH /aws1/cx_iamcredrptexpiredex.
        MESSAGE 'Credential report expired.' TYPE 'E'.
      CATCH /aws1/cx_iamcredrptnotreadyex.
        MESSAGE 'Credential report not ready.' TYPE 'E'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when getting credential report.' TYPE 'E'.
    ENDTRY.
```

- For API details, see GetCredentialReport in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetGroup` with a CLI

The following code examples show how to use `GetGroup`.

CLI

AWS CLI

To get an IAM group

The following `get-group` example returns details about the IAM group `Admins`.

```
aws iam get-group \
    --group-name Admins
```

Output:

```
{
    "Group": {
        "Path": "/",
```

```
        "CreateDate": "2015-06-16T19:41:48Z",
        "GroupId": "AIDGPMS9RO4H3FEXAMPLE",
        "Arn": "arn:aws:iam::123456789012:group/Admins",
        "GroupName": "Admins"
    },
    "Users": []
}
```

For more information, see IAM Identities (users, user groups, and roles) in the *AWS IAM User Guide*.

- For API details, see GetGroup in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details about the IAM group `Testers`, including a collection of all the IAM users that belong to the group.

```
$results = Get-IAMGroup -GroupName "Testers"
$results
```

Output:

```
Group                                    IsTruncated           Marker
     Users
-----                                    -----------           ------
     -----
Amazon.IdentityManagement.Model.Group    False
     {Theresa, David}
```

```
$results.Group
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Testers
CreateDate : 12/10/2014 3:39:11 PM
GroupId    : 3RHNZZGQJ7QHMAEXAMPLE1
GroupName  : Testers
```

```
Path       : /
```

```
$results.Users
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Theresa
CreateDate       : 12/10/2014 3:39:27 PM
PasswordLastUsed : 1/1/0001 12:00:00 AM
Path             : /
UserId           : 4OSVDDJJTF4XEEXAMPLE2
UserName         : Theresa

Arn              : arn:aws:iam::123456789012:user/David
CreateDate       : 12/10/2014 3:39:27 PM
PasswordLastUsed : 3/19/2015 8:44:04 AM
Path             : /
UserId           : Y4FKWQCXTA52QEXAMPLE3
UserName         : David
```

- For API details, see GetGroup in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details about the IAM group `Testers`, including a collection of all the IAM users that belong to the group.

```
$results = Get-IAMGroup -GroupName "Testers"
$results
```

Output:

```
Group                                    IsTruncated           Marker
     Users
-----                                    -----------           ------
     -----
Amazon.IdentityManagement.Model.Group    False
     {Theresa, David}
```

```
$results.Group
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Testers
CreateDate : 12/10/2014 3:39:11 PM
GroupId    : 3RHNZZGQJ7QHMAEXAMPLE1
GroupName  : Testers
Path       : /
```

```
$results.Users
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Theresa
CreateDate       : 12/10/2014 3:39:27 PM
PasswordLastUsed : 1/1/0001 12:00:00 AM
Path             : /
UserId           : 4OSVDDJJTF4XEEXAMPLE2
UserName         : Theresa

Arn              : arn:aws:iam::123456789012:user/David
CreateDate       : 12/10/2014 3:39:27 PM
PasswordLastUsed : 3/19/2015 8:44:04 AM
Path             : /
UserId           : Y4FKWQCXTA52QEXAMPLE3
UserName         : David
```

- For API details, see GetGroup in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetGroupPolicy` with a CLI

The following code examples show how to use `GetGroupPolicy`.

CLI

AWS CLI

To get information about a policy attached to an IAM group

The following `get-group-policy` example gets information about the specified policy attached to the group named `Test-Group`.

```
aws iam get-group-policy \
    --group-name Test-Group \
    --policy-name S3-ReadOnly-Policy
```

Output:

```
{
    "GroupName": "Test-Group",
    "PolicyDocument": {
        "Statement": [
            {
                "Action": [
                    "s3:Get*",
                    "s3:List*"
                ],
                "Resource": "*",
                "Effect": "Allow"
            }
        ]
    },
    "PolicyName": "S3-ReadOnly-Policy"
}
```

For more information, see Managing IAM policies in the *AWS IAM User Guide*.

- For API details, see GetGroupPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details about the embedded inline policy named `PowerUserAccess-Testers` for the group `Testers`. The `PolicyDocument` property is URL encoded. It is decoded in this example with the `UrlDecode` .NET method.

```
$results = Get-IAMGroupPolicy -GroupName Testers -PolicyName PowerUserAccess-
Testers
$results
```

Output:

```
GroupName     PolicyDocument
 PolicyName
---------     --------------
 ----------
Testers       %7B%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C%0A%20...
 PowerUserAccess-Testers

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
[System.Web.HttpUtility]::UrlDecode($results.PolicyDocument)
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Resource": [
        "arn:aws:ec2:us-east-1:555555555555:instance/i-b188560f"
      ]
    }
  ]
}
```

- For API details, see GetGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details about the embedded inline policy named `PowerUserAccess-Testers` for the group `Testers`. The `PolicyDocument` property is URL encoded. It is decoded in this example with the `UrlDecode` .NET method.

```
$results = Get-IAMGroupPolicy -GroupName Testers -PolicyName PowerUserAccess-
Testers
$results
```

Output:

```
GroupName     PolicyDocument
 PolicyName
```

```
---------     --------------
 ----------
Testers       %7B%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C%0A%20...
 PowerUserAccess-Testers

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
[System.Web.HttpUtility]::UrlDecode($results.PolicyDocument)
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Resource": [
        "arn:aws:ec2:us-east-1:555555555555:instance/i-b188560f"
      ]
    }
  ]
}
```

- For API details, see GetGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetInstanceProfile` with a CLI

The following code examples show how to use `GetInstanceProfile`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Getting started with managed streaming

CLI

AWS CLI

To get information about an instance profile

The following `get-instance-profile` example gets information about the instance profile named `ExampleInstanceProfile`.

```
aws iam get-instance-profile \
    --instance-profile-name ExampleInstanceProfile
```

Output:

```
{
    "InstanceProfile": {
        "InstanceProfileId": "AID2MAB8DPLSRHEXAMPLE",
        "Roles": [
            {
                "AssumeRolePolicyDocument": "<URL-encoded-JSON>",
                "RoleId": "AIDGPMS9RO4H3FEXAMPLE",
                "CreateDate": "2013-01-09T06:33:26Z",
                "RoleName": "Test-Role",
                "Path": "/",
                "Arn": "arn:aws:iam::123456789012:role/Test-Role"
            }
        ],
        "CreateDate": "2013-06-12T23:52:02Z",
        "InstanceProfileName": "ExampleInstanceProfile",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:instance-profile/
ExampleInstanceProfile"
    }
}
```

For more information, see Using instance profiles in the *AWS IAM User Guide*.

- For API details, see GetInstanceProfile in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details of the instance profile named `ec2instancerole` that is defined in the current AWS account.

```
Get-IAMInstanceProfile -InstanceProfileName ec2instancerole
```

Output:

```
Arn                 : arn:aws:iam::123456789012:instance-profile/ec2instancerole
CreateDate          : 2/17/2015 2:49:04 PM
InstanceProfileId   : HH36PTZQJUR32EXAMPLE1
InstanceProfileName : ec2instancerole
Path                : /
Roles               : {ec2instancerole}
```

- For API details, see GetInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details of the instance profile named `ec2instancerole` that is defined in the current AWS account.

```
Get-IAMInstanceProfile -InstanceProfileName ec2instancerole
```

Output:

```
Arn                 : arn:aws:iam::123456789012:instance-profile/ec2instancerole
CreateDate          : 2/17/2015 2:49:04 PM
InstanceProfileId   : HH36PTZQJUR32EXAMPLE1
InstanceProfileName : ec2instancerole
Path                : /
Roles               : {ec2instancerole}
```

- For API details, see GetInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetLoginProfile` with a CLI

The following code examples show how to use `GetLoginProfile`.

CLI

AWS CLI

To get password information for an IAM user

The following `get-login-profile` example gets information about the password for the IAM user named Bob.

```
aws iam get-login-profile \
    --user-name Bob
```

Output:

```
{
    "LoginProfile": {
        "UserName": "Bob",
        "CreateDate": "2012-09-21T23:03:39Z"
    }
}
```

The `get-login-profile` command can be used to verify that an IAM user has a password. The command returns a `NoSuchEntity` error if no password is defined for the user.

You cannot view a password using this command. If the password is lost, you can reset the password (`update-login-profile`) for the user. Alternatively, you can delete the login profile (`delete-login-profile`) for the user and then create a new one (`create-login-profile`).

For more information, see Managing passwords for IAM users in the *AWS IAM User Guide*.

- For API details, see GetLoginProfile in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns the password creation date and whether a password reset is required for the IAM user `David`.

```
Get-IAMLoginProfile -UserName David
```

Output:

```
CreateDate                 PasswordResetRequired              UserName
----------                 ---------------------              --------
```

```
12/10/2014 3:39:44 PM        False                                  David
```

- For API details, see GetLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns the password creation date and whether a password reset is required for the IAM user `David`.

```
Get-IAMLoginProfile -UserName David
```

Output:

```
CreateDate                  PasswordResetRequired                UserName
----------                  ---------------------                --------
12/10/2014 3:39:44 PM       False                                David
```

- For API details, see GetLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetOpenIdConnectProvider` with a CLI

The following code examples show how to use `GetOpenIdConnectProvider`.

CLI

AWS CLI

To return information about the specified OpenID Connect provider

The following `get-open-id-connect-provider` example returns details about the OpenID Connect provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/server.example.com`.

```
aws iam get-open-id-connect-provider \
    --open-id-connect-provider-arn arn:aws:iam::123456789012:oidc-provider/
server.example.com
```

Output:

```
{
    "Url": "server.example.com",
    "CreateDate": "2015-06-16T19:41:48Z",
    "ThumbprintList": [
        "12345abcdefghijk67890lmnopqrst987example"
    ],
    "ClientIDList": [
        "example-application-ID"
    ]
}
```

For more information, see Creating OpenID Connect (OIDC) identity providers in the *AWS IAM User Guide*.

- For API details, see GetOpenIdConnectProvider in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details about the OpenID Connect provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/accounts.google.com`. The `ClientIDList` property is a collection that contains all the Client IDs defined for this provider.

```
Get-IAMOpenIDConnectProvider -OpenIDConnectProviderArn
 arn:aws:iam::123456789012:oidc-provider/oidc.example.com
```

Output:

```
ClientIDList         CreateDate                ThumbprintList
           Url
------------         ----------                --------------
           ---
{MyOIDCApp}          2/3/2015 3:00:30 PM
 {12345abcdefghijk67890lmnopqrst98765uvwxy}   oidc.example.com
```

- For API details, see GetOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details about the OpenID Connect provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/accounts.google.com`. The `ClientIDList` property is a collection that contains all the Client IDs defined for this provider.

```
Get-IAMOpenIDConnectProvider -OpenIDConnectProviderArn
 arn:aws:iam::123456789012:oidc-provider/oidc.example.com
```

Output:

```
ClientIDList         CreateDate                ThumbprintList
           Url
------------         ----------                --------------
           ---
{MyOIDCApp}          2/3/2015 3:00:30 PM
 {12345abcdefghijk67890lmnopqrst98765uvwxy}   oidc.example.com
```

- For API details, see GetOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetPolicy` with an AWS SDK or CLI

The following code examples show how to use `GetPolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Getting started with managed streaming
- Work with the IAM Policy Builder API

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</
param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
 { PolicyArn = policyArn });
        return response.Policy;
    }
```

- For API details, see GetPolicy in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::getPolicy(const Aws::String &policyArn,
```

```
                            const Aws::Client::ClientConfiguration &clientConfig)
 {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::GetPolicyRequest request;
    request.SetPolicyArn(policyArn);

    auto outcome = iam.GetPolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error getting policy " << policyArn << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        const auto &policy = outcome.GetResult().GetPolicy();
        std::cout << "Name: " << policy.GetPolicyName() << std::endl <<
                  "ID: " << policy.GetPolicyId() << std::endl << "Arn: " <<
                  policy.GetArn() << std::endl << "Description: " <<
                  policy.GetDescription() << std::endl << "CreateDate: " <<

 policy.GetCreateDate().ToGmtString(Aws::Utils::DateFormat::ISO_8601)
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see GetPolicy in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To retrieve information about the specified managed policy

The following `get-policy` example returns details about the managed policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
aws iam get-policy \
    --policy-arn arn:aws:iam::123456789012:policy/MySamplePolicy
```

Output:

```
{
    "Policy": {
        "PolicyName": "MySamplePolicy",
        "CreateDate": "2015-06-17T19:23:32Z",
        "AttachmentCount": 0,
        "IsAttachable": true,
        "PolicyId": "Z27SI6FQMGNQ2EXAMPLE1",
        "DefaultVersionId": "v1",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:policy/MySamplePolicy",
        "UpdateDate": "2015-06-17T19:23:32Z"
    }
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see GetPolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
```

```
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(ctx context.Context, policyArn string)
 (*types.Policy, error) {
 var policy *types.Policy
 result, err := wrapper.IamClient.GetPolicy(ctx, &iam.GetPolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}
```

- For API details, see GetPolicy in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get the policy.

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetPolicy in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
```

```
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetPolicy in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getIAMPolicy(policyArnVal: String?) {
    val request =
        GetPolicyRequest {
            policyArn = policyArnVal
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.getPolicy(request)
        println("Successfully retrieved policy ${response.policy?.policyName}")
    }
}
```

- For API details, see GetPolicy in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function getPolicy($policyArn)
    {
        return $this->customWaiter(function () use ($policyArn) {
            return $this->iamClient->getPolicy(['PolicyArn' => $policyArn]);
        });
    }
```

- For API details, see GetPolicy in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details about the managed policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
```

Output:

```
Arn              : arn:aws:iam::aws:policy/MySamplePolicy
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:08 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
```

```
Path             : /
PolicyId         : Z27SI6FQMGNQ2EXAMPLE1
PolicyName       : MySamplePolicy
UpdateDate       : 2/6/2015 10:40:08 AM
```

- For API details, see GetPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details about the managed policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
Get-IAMPolicy -PolicyArn arn:aws:iam::123456789012:policy/MySamplePolicy
```

Output:

```
Arn              : arn:aws:iam::aws:policy/MySamplePolicy
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:08 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : Z27SI6FQMGNQ2EXAMPLE1
PolicyName       : MySamplePolicy
UpdateDate       : 2/6/2015 10:40:08 AM
```

- For API details, see GetPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_default_policy_statement(policy_arn):
    """
```

```
    Gets the statement of the default version of the specified policy.

    :param policy_arn: The ARN of the policy to look up.
    :return: The statement of the default policy version.
    """
    try:
        policy = iam.Policy(policy_arn)
        # To get an attribute of a policy, the SDK first calls get_policy.
        policy_doc = policy.default_version.document
        policy_statement = policy_doc.get("Statement", None)
        logger.info("Got default policy doc for %s.", policy.policy_name)
        logger.info(policy_doc)
    except ClientError:
        logger.exception("Couldn't get default policy statement for %s.",
policy_arn)
        raise
    else:
        return policy_statement
```

- For API details, see GetPolicy in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
 # Fetches an IAM policy by its ARN
 # @param policy_arn [String] the ARN of the IAM policy to retrieve
 # @return [Aws::IAM::Types::GetPolicyResponse] the policy object if found
 def get_policy(policy_arn)
   response = @iam_client.get_policy(policy_arn: policy_arn)
   policy = response.policy
   @logger.info("Got policy '#{policy.policy_name}'. Its ID is:
#{policy.policy_id}.")
```

```
    policy
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Couldn't get policy '#{policy_arn}'. The policy does not
exist.")
    raise
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't get policy '#{policy_arn}'. Here's why: #{e.code}:
#{e.message}")
    raise
  end
```

- For API details, see GetPolicy in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->getpolicy( iv_policyarn = iv_policy_arn ).
        MESSAGE 'Retrieved policy information.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Policy does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see GetPolicy in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func getPolicy(arn: String) async throws -> IAMClientTypes.Policy {
        let input = GetPolicyInput(
            policyArn: arn
        )
        do {
            let output = try await client.getPolicy(input: input)
            guard let policy = output.policy else {
                throw ServiceHandlerError.noSuchPolicy
            }
            return policy
        } catch {
            print("ERROR: getPolicy:", dump(error))
            throw error
        }
    }
```

- For API details, see GetPolicy in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetPolicyVersion` with an AWS SDK or CLI

The following code examples show how to use `GetPolicyVersion`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Manage policies
- Work with the IAM Policy Builder API

CLI

AWS CLI

To retrieve information about the specified version of the specified managed policy

The following `get-policy-version` example returns the policy document for the v2 version of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyManagedPolicy`.

```
aws iam get-policy-version \
    --policy-arn arn:aws:iam::123456789012:policy/MyPolicy \
    --version-id v2
```

Output:

```
{
    "PolicyVersion": {
        "Document": {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": "iam:*",
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v2",
        "IsDefaultVersion": true,
        "CreateDate": "2023-04-11T00:22:54+00:00"
    }
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see GetPolicyVersion in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns the policy document for the v2 version of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyManagedPolicy`. The policy document in the `Document` property is URL encoded and is decoded in this example with the `UrlDecode` .NET method.

```
$results = Get-IAMPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/
MyManagedPolicy -VersionId v2
$results
```

Output:

```
CreateDate             Document
 IsDefaultVersion     VersionId
----------             --------
 ----------------     ---------
2/12/2015 9:39:53 AM   %7B%0A%20%20%22Version%22%3A%20%222012-10...    True
           v2

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
$policy = [System.Web.HttpUtility]::UrlDecode($results.Document)
$policy
{
  "Version":"2012-10-17",
  "Statement":
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Resource": [
        "arn:aws:ec2:us-east-1:555555555555:instance/i-b188560f"
      ]
    }
}
```

- For API details, see GetPolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns the policy document for the v2 version of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyManagedPolicy`. The policy document in the `Document` property is URL encoded and is decoded in this example with the `UrlDecode` .NET method.

```
$results = Get-IAMPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/
MyManagedPolicy -VersionId v2
$results
```

Output:

```
CreateDate             Document
 IsDefaultVersion     VersionId
----------             --------
 ----------------     ---------
2/12/2015 9:39:53 AM   %7B%0A%20%20%22Version%22%3A%20%222012-10...    True
           v2

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
$policy = [System.Web.HttpUtility]::UrlDecode($results.Document)
$policy
{
  "Version":"2012-10-17",
  "Statement":
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Resource": [
        "arn:aws:ec2:us-east-1:555555555555:instance/i-b188560f"
      ]
    }
}
```

- For API details, see GetPolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_default_policy_statement(policy_arn):
    """
    Gets the statement of the default version of the specified policy.

    :param policy_arn: The ARN of the policy to look up.
    :return: The statement of the default policy version.
    """
    try:
        policy = iam.Policy(policy_arn)
        # To get an attribute of a policy, the SDK first calls get_policy.
        policy_doc = policy.default_version.document
        policy_statement = policy_doc.get("Statement", None)
        logger.info("Got default policy doc for %s.", policy.policy_name)
        logger.info(policy_doc)
    except ClientError:
        logger.exception("Couldn't get default policy statement for %s.",
 policy_arn)
        raise
    else:
        return policy_statement
```

- For API details, see GetPolicyVersion in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->getpolicyversion(
      iv_policyarn = iv_policy_arn
      iv_versionid = iv_version_id ).
    MESSAGE 'Retrieved policy version information.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'Policy or version does not exist.' TYPE 'E'.
  CATCH /aws1/cx_iaminvalidinputex.
    MESSAGE 'Invalid input provided.' TYPE 'E'.
ENDTRY.
```

- For API details, see GetPolicyVersion in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetRole` with an AWS SDK or CLI

The following code examples show how to use `GetRole`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Configure container service connectivity
- Create a container task for the serverless launch type
- Creating a container service for virtual machine instances
- Get started with serverless data warehouses

- Getting started with internet of things device protection
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with provisioned data warehouse clusters
- Run CPU stress tests on virtual machine instances using fault injection

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Get information about an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to retrieve information
/// for.</param>
/// <returns>The IAM role that was retrieved.</returns>
public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
    {
        RoleName = roleName,
    });

    return response.Role;
}
```

- For API details, see GetRole in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To get information about an IAM role

The following `get-role` example gets information about the role named `Test-Role`.

```
aws iam get-role \
    --role-name Test-Role
```

Output:

```
{
    "Role": {
        "Description": "Test Role",
        "AssumeRolePolicyDocument":"<URL-encoded-JSON>",
        "MaxSessionDuration": 3600,
        "RoleId": "AROA1234567890EXAMPLE",
        "CreateDate": "2019-11-13T16:45:56Z",
        "RoleName": "Test-Role",
        "Path": "/",
        "RoleLastUsed": {
            "Region": "us-west-2",
            "LastUsedDate": "2019-11-13T17:14:00Z"
        },
        "Arn": "arn:aws:iam::123456789012:role/Test-Role"
    }
}
```

The command displays the trust policy attached to the role. To list the permissions policies attached to a role, use the `list-role-policies` command.

For more information, see Creating IAM roles in the *AWS IAM User Guide*.

- For API details, see GetRole in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(ctx context.Context, roleName string)
 (*types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.GetRole(ctx,
  &iam.GetRoleInput{RoleName: aws.String(roleName)})
 if err != nil {
  log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
```

```
}
```

- For API details, see GetRole in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get the role.

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- For API details, see GetRole in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function getRole($roleName)
    {
        return $this->customWaiter(function () use ($roleName) {
            return $this->iamClient->getRole(['RoleName' => $roleName]);
        });
    }
```

- For API details, see GetRole in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns the details of the `lamda_exec_role`. It includes the trust policy document that specifies who can assume this role. The policy document is URL encoded and can be decoded using the .NET `UrlDecode` method. In this example, the original policy had all white space removed before it was uploaded to the policy. To see the permissions policy documents that determine what someone who assumes the role can do, use the `Get-IAMRolePolicy` for inline policies, and `Get-IAMPolicyVersion` for attached managed policies.

```
$results = Get-IamRole -RoleName lambda_exec_role
$results | Format-List
```

Output:

```
Arn                      : arn:aws:iam::123456789012:role/lambda_exec_role
AssumeRolePolicyDocument : %7B%22Version%22%3A%222012-10-17%22%2C%22Statement
%22%3A%5B%7B%22Sid%22
                           %3A%22%22%2C%22Effect%22%3A%22Allow%22%2C%22Principal
%22%3A%7B%22Service
                           %22%3A%22lambda.amazonaws.com%22%7D%2C%22Action%22%3A
%22sts%3AAssumeRole
                           %22%7D%5D%7D
CreateDate               : 4/2/2015 9:16:11 AM
Path                     : /
RoleId                   : 2YBIKAIBHNKB4EXAMPLE1
RoleName                 : lambda_exec_role
```

```
$policy = [System.Web.HttpUtility]::UrlDecode($results.AssumeRolePolicyDocument)
$policy
```

Output:

```
{"Version":"2012-10-17",       "Statement":
[{"Sid":"","Effect":"Allow","Principal":
{"Service":"lambda.amazonaws.com"},"Action":"sts:AssumeRole"}]}
```

- For API details, see GetRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns the details of the `lamda_exec_role`. It includes the trust policy document that specifies who can assume this role. The policy document is URL encoded and can be decoded using the .NET `UrlDecode` method. In this example, the original policy had all white space removed before it was uploaded to the policy. To see the permissions policy documents that determine what someone who assumes the role can do, use the `Get-IAMRolePolicy` for inline policies, and `Get-IAMPolicyVersion` for attached managed policies.

```
$results = Get-IamRole -RoleName lambda_exec_role
$results | Format-List
```

Output:

```
Arn                      : arn:aws:iam::123456789012:role/lambda_exec_role
```

```
AssumeRolePolicyDocument : %7B%22Version%22%3A%222012-10-17%22%2C%22Statement
%22%3A%5B%7B%22Sid%22
                           %3A%22%22%2C%22Effect%22%3A%22Allow%22%2C%22Principal
%22%3A%7B%22Service
                           %22%3A%22lambda.amazonaws.com%22%7D%2C%22Action%22%3A
%22sts%3AAssumeRole
                           %22%7D%5D%7D
CreateDate               : 4/2/2015 9:16:11 AM
Path                     : /
RoleId                   : 2YBIKAIBHNKB4EXAMPLE1
RoleName                 : lambda_exec_role
```

```
$policy = [System.Web.HttpUtility]::UrlDecode($results.AssumeRolePolicyDocument)
$policy
```

Output:

```
{"Version":"2012-10-17",       "Statement":
[{"Sid":"","Effect":"Allow","Principal":
{"Service":"lambda.amazonaws.com"},"Action":"sts:AssumeRole"}]}
```

- For API details, see GetRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_role(role_name):
    """
    Gets a role by name.

    :param role_name: The name of the role to retrieve.
    :return: The specified role.
    """
```

```
    try:
        role = iam.Role(role_name)
        role.load()  # calls GetRole to load attributes
        logger.info("Got role with arn %s.", role.arn)
    except ClientError:
        logger.exception("Couldn't get role named %s.", role_name)
        raise
    else:
        return role
```

- For API details, see GetRole in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Gets data about a role.
#
# @param name [String] The name of the role to look up.
# @return [Aws::IAM::Role] The retrieved role.
def get_role(name)
  role = @iam_client.get_role({
                                role_name: name
                              }).role
  puts("Got data for role '#{role.role_name}'. Its ARN is '#{role.arn}'.")
rescue Aws::Errors::ServiceError => e
  puts("Couldn't get data for role '#{name}' Here's why:")
  puts("\t#{e.code}: #{e.message}")
  raise
else
  role
end
```

- For API details, see GetRole in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn get_role(
    client: &iamClient,
    role_name: String,
) -> Result<GetRoleOutput, SdkError<GetRoleError>> {
    let response = client.get_role().role_name(role_name).send().await?;
    Ok(response)
}
```

- For API details, see GetRole in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->getrole( iv_rolename = iv_role_name ).
        MESSAGE 'Retrieved role information.' TYPE 'I'.
```

```
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Role does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see GetRole in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func getRole(name: String) async throws -> IAMClientTypes.Role {
        let input = GetRoleInput(
            roleName: name
        )
        do {
            let output = try await client.getRole(input: input)
            guard let role = output.role else {
                throw ServiceHandlerError.noSuchRole
            }
            return role
        } catch {
            print("ERROR: getRole:", dump(error))
            throw error
        }
    }
```

- For API details, see GetRole in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetRolePolicy` with a CLI

The following code examples show how to use `GetRolePolicy`.

CLI

AWS CLI

To get information about a policy attached to an IAM role

The following `get-role-policy` example gets information about the specified policy attached to the role named `Test-Role`.

```
aws iam get-role-policy \
    --role-name Test-Role \
    --policy-name ExamplePolicy
```

Output:

```
{
    "RoleName": "Test-Role",
    "PolicyDocument": {
        "Statement": [
            {
                "Action": [
                    "s3:ListBucket",
                    "s3:Put*",
                    "s3:Get*",
                    "s3:*MultipartUpload*"
                ],
                "Resource": "*",
                "Effect": "Allow",
                "Sid": "1"
            }
        ]
    },
    "PolicyName": "ExamplePolicy"
}
```

For more information, see Creating IAM roles in the *AWS IAM User Guide*.

- For API details, see GetRolePolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns the permissions policy document for the policy named oneClick_lambda_exec_role_policy that is embedded in the IAM role lamda_exec_role. The resulting policy document is URL encoded. It is decoded in this example with the UrlDecode .NET method.

```
$results = Get-IAMRolePolicy -RoleName lambda_exec_role -PolicyName
 oneClick_lambda_exec_role_policy
$results
```

Output:

```
PolicyDocument                                          PolicyName
              UserName
--------------                                          ----------
              --------
%7B%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C%...
 oneClick_lambda_exec_role_policy     lambda_exec_role
```

```
[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
[System.Web.HttpUtility]::UrlDecode($results.PolicyDocument)
```

Output:

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:*"
      ],
      "Resource": "arn:aws:logs:us-east-1:555555555555:log-group:/aws/lambda/aws-
example-function:*"
```

```
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    }
  ]
}
```

- For API details, see GetRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns the permissions policy document for the policy named `oneClick_lambda_exec_role_policy` that is embedded in the IAM role `lamda_exec_role`. The resulting policy document is URL encoded. It is decoded in this example with the `UrlDecode` .NET method.

```
$results = Get-IAMRolePolicy -RoleName lambda_exec_role -PolicyName
 oneClick_lambda_exec_role_policy
$results
```

Output:

```
PolicyDocument                                          PolicyName
              UserName
--------------                                          ----------
              --------
%7B%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C%...
 oneClick_lambda_exec_role_policy     lambda_exec_role
```

```
[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
[System.Web.HttpUtility]::UrlDecode($results.PolicyDocument)
```

Output:

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:*"
      ],
      "Resource": "arn:aws:logs:us-east-1:555555555555:log-group:/aws/lambda/aws-
example-function:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    }
  ]
}
```

- For API details, see GetRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetSamlProvider` with a CLI

The following code examples show how to use `GetSamlProvider`.

CLI

AWS CLI

To retrieve the SAML provider metadocument

The following `get-saml-provider` example retrieves the details about the SAML 2.0 provider whose ARN is `arn:aws:iam::123456789012:saml-provider/SAMLADFS`. The

response includes the metadata document that you got from the identity provider to create the AWS SAML provider entity as well as the creation and expiration dates.

```
aws iam get-saml-provider \
    --saml-provider-arn arn:aws:iam::123456789012:saml-provider/SAMLADFS
```

Output:

```
{
    "SAMLMetadataDocument": "...SAMLMetadataDocument-XML...",
    "CreateDate": "2017-03-06T22:29:46+00:00",
    "ValidUntil": "2117-03-06T22:29:46.433000+00:00",
    "Tags": [
        {
            "Key": "DeptID",
            "Value": "123456"
        },
        {
            "Key": "Department",
            "Value": "Accounting"
        }
    ]
}
```

For more information, see Creating IAM SAML identity providers in the *AWS IAM User Guide*.

- For API details, see GetSamlProvider in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves the details about the SAML 2.0 provider whose ARM is arn:aws:iam::123456789012:saml-provider/SAMLADFS. The response includes the metadata document that you got from the identity provider to create the AWS SAML provider entity as well as the creation and expiration dates.

```
Get-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/
SAMLADFS
```

Output:

```
CreateDate                 SAMLMetadataDocument
        ValidUntil
----------                 --------------------
        ----------
12/23/2014 12:16:55 PM    <EntityDescriptor ID="_12345678-1234-5678-9012-
example1...    12/23/2114 12:16:54 PM
```

- For API details, see GetSamlProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves the details about the SAML 2.0 provider whose ARM is arn:aws:iam::123456789012:saml-provider/SAMLADFS. The response includes the metadata document that you got from the identity provider to create the AWS SAML provider entity as well as the creation and expiration dates.

```
Get-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/
SAMLADFS
```

Output:

```
CreateDate                 SAMLMetadataDocument
        ValidUntil
----------                 --------------------
        ----------
12/23/2014 12:16:55 PM    <EntityDescriptor ID="_12345678-1234-5678-9012-
example1...    12/23/2114 12:16:54 PM
```

- For API details, see GetSamlProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetServerCertificate` with an AWS SDK or CLI

The following code examples show how to use `GetServerCertificate`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::getServerCertificate(const Aws::String &certificateName,
                                       const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::GetServerCertificateRequest request;
    request.SetServerCertificateName(certificateName);

    auto outcome = iam.GetServerCertificate(request);
    bool result = true;
    if (!outcome.IsSuccess()) {
        if (outcome.GetError().GetErrorType() !=
 Aws::IAM::IAMErrors::NO_SUCH_ENTITY) {
            std::cerr << "Error getting server certificate " << certificateName
 <<
                      ": " << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Certificate '" << certificateName
                      << "' not found." << std::endl;
        }
    }
    else {
        const auto &certificate = outcome.GetResult().GetServerCertificate();
        std::cout << "Name: " <<

 certificate.GetServerCertificateMetadata().GetServerCertificateName()
                  << std::endl << "Body: " << certificate.GetCertificateBody() <<
                  std::endl << "Chain: " << certificate.GetCertificateChain() <<
                  std::endl;
    }
```

```
    return result;
}
```

- For API details, see GetServerCertificate in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To get details about a server certificate in your AWS account

The following `get-server-certificate` example retrieves all of the details about the specified server certificate in your AWS account.

```
aws iam get-server-certificate \
    --server-certificate-name myUpdatedServerCertificate
```

Output:

```
{
    "ServerCertificate": {
        "ServerCertificateMetadata": {
            "Path": "/",
            "ServerCertificateName": "myUpdatedServerCertificate",
            "ServerCertificateId": "ASCAEXAMPLE123EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:server-certificate/
myUpdatedServerCertificate",
            "UploadDate": "2019-04-22T21:13:44+00:00",
            "Expiration": "2019-10-15T22:23:16+00:00"
        },
        "CertificateBody": "-----BEGIN CERTIFICATE-----
\nMIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
\nVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6\nb24xFDASBgNVBAsTC0lBT
\nBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN
\nMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD
\nVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z
\nb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt
\nYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ
\n21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
\nrDHudUZg3qX4waLG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
\nIbb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4\nnUhVVxYUntneD9+h8Mg9q
```

```
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
\nFFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
\nNYiytVbZPQUQ5Yaxu2jXnimvrszlaEXAMPLE=\n-----END CERTIFICATE-----",
        "CertificateChain": "-----BEGIN CERTIFICATE-----
\nMIICiTCCAfICCQD6md7oRw0uXOjANBgkqhkiG9w0BAqQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBM
+a4GmWIgWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/
f0wYK8mh9TrDHudUZg3qX4waLG5M43q7Wgc/
MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gjpEIbb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQ
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FlkbFFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD
\n-----END CERTIFICATE-----"
    }
}
```

To list the server certificates available in your AWS account, use the `list-server-certificates` command.

For more information, see Managing server certificates in IAM in the *AWS IAM User Guide*.

- For API details, see GetServerCertificate in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get a server certificate.

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
```

```
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetServerCertificate in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- For more information, see AWS SDK for JavaScript Developer Guide.

- For API details, see GetServerCertificate in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves details about the server certificate named `MyServerCertificate`. You can find the certificate details in the `CertificateBody` and `ServerCertificateMetadata` properties.

```
$result = Get-IAMServerCertificate -ServerCertificateName MyServerCertificate
$result | format-list
```

Output:

```
CertificateBody           : -----BEGIN CERTIFICATE-----

 MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC

 VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6

 b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd

 BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN

 MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD

 VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z

 b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt

 YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ
                            21uUSfwfEvySWtC2XADZ4nB
+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
                            rDHudUZg3qX4waLG5M43q7Wgc/
MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE

 Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
                            nUhVVxYUntneD9+h8Mg9q6q
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb

 FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
```

```
                            NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
                            -----END CERTIFICATE-----
CertificateChain          :
ServerCertificateMetadata :
 Amazon.IdentityManagement.Model.ServerCertificateMetadata
```

```
$result.ServerCertificateMetadata
```

Output:

```
Arn                   : arn:aws:iam::123456789012:server-certificate/Org1/Org2/
MyServerCertificate
Expiration            : 1/14/2018 9:52:36 AM
Path                  : /Org1/Org2/
ServerCertificateId   : ASCAJIFEXAMPLE17HQZYW
ServerCertificateName : MyServerCertificate
UploadDate            : 4/21/2015 11:14:16 AM
```

- For API details, see GetServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves details about the server certificate named `MyServerCertificate`. You can find the certificate details in the `CertificateBody` and `ServerCertificateMetadata` properties.

```
$result = Get-IAMServerCertificate -ServerCertificateName MyServerCertificate
$result | format-list
```

Output:

```
CertificateBody           : -----BEGIN CERTIFICATE-----

 MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC

 VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6

 b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd

 BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN

 MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD
```

```
 VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z

 b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt

 YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ
                            21uUSfwfEvySWtC2XADZ4nB
+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
                            rDHudUZg3qX4waLG5M43q7Wgc/
MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE

 Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
                            nUhVVxYUntneD9+h8Mg9q6q
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb

 FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
                            NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
                            -----END CERTIFICATE-----
CertificateChain          :
ServerCertificateMetadata :
 Amazon.IdentityManagement.Model.ServerCertificateMetadata
```

```
$result.ServerCertificateMetadata
```

Output:

```
Arn                   : arn:aws:iam::123456789012:server-certificate/Org1/Org2/
MyServerCertificate
Expiration            : 1/14/2018 9:52:36 AM
Path                  : /Org1/Org2/
ServerCertificateId   : ASCAJIFEXAMPLE17HQZYW
ServerCertificateName : MyServerCertificate
UploadDate            : 4/21/2015 11:14:16 AM
```

- For API details, see GetServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetServiceLastAccessedDetails` with a CLI

The following code examples show how to use `GetServiceLastAccessedDetails`.

CLI

AWS CLI

To retrieve a service access report

The following `get-service-last-accessed-details` example retrieves a previously generated report that lists the services accessed by IAM entities. To generate a report, use the `generate-service-last-accessed-details` command.

```
aws iam get-service-last-accessed-details \
    --job-id a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

Output:

```
{
    "JobStatus": "COMPLETED",
    "JobCreationDate": "2019-10-01T03:50:35.929Z",
    "ServicesLastAccessed": [
        {
            "ServiceName": "AWS Lambda",
            "LastAuthenticated": "2019-09-30T23:02:00Z",
            "ServiceNamespace": "lambda",
            "LastAuthenticatedEntity": "arn:aws:iam::123456789012:user/admin",
            "TotalAuthenticatedEntities": 6
        }
    ],
    "IsTruncated": false
}
```

For more information, see Refining permissions in AWS using last accessed information in the *AWS IAM User Guide*.

- For API details, see GetServiceLastAccessedDetails in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example provides details of the service last accessed by the IAM entity(user, group, role or policy) associated in Request call.

```
Request-IAMServiceLastAccessedDetail -Arn arn:aws:iam::123456789012:user/TestUser
```

Output:

```
f0b7a819-eab0-929b-dc26-ca598911cb9f
```

```
Get-IAMServiceLastAccessedDetail -JobId f0b7a819-eab0-929b-dc26-ca598911cb9f
```

- For API details, see GetServiceLastAccessedDetails in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example provides details of the service last accessed by the IAM entity(user, group, role or policy) associated in Request call.

```
Request-IAMServiceLastAccessedDetail -Arn arn:aws:iam::123456789012:user/TestUser
```

Output:

```
f0b7a819-eab0-929b-dc26-ca598911cb9f
```

```
Get-IAMServiceLastAccessedDetail -JobId f0b7a819-eab0-929b-dc26-ca598911cb9f
```

- For API details, see GetServiceLastAccessedDetails in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetServiceLastAccessedDetailsWithEntities` with a CLI

The following code examples show how to use `GetServiceLastAccessedDetailsWithEntities`.

CLI

AWS CLI

To retrieve a service access report with details for a service

The following `get-service-last-accessed-details-with-entities` example retrieves a report that contains details about IAM users and other entities that accessed the specified service. To generate a report, use the `generate-service-last-accessed-details` command. To get a list of services accessed with namespaces, use `get-service-last-accessed-details`.

```
aws iam get-service-last-accessed-details-with-entities \
    --job-id a1b2c3d4-5678-90ab-cdef-EXAMPLE22222 \
    --service-namespace lambda
```

Output:

```
{
    "JobStatus": "COMPLETED",
    "JobCreationDate": "2019-10-01T03:55:41.756Z",
    "JobCompletionDate": "2019-10-01T03:55:42.533Z",
    "EntityDetailsList": [
        {
            "EntityInfo": {
                "Arn": "arn:aws:iam::123456789012:user/admin",
                "Name": "admin",
                "Type": "USER",
                "Id": "AIDAIO2XMPLENQEXAMPLE",
                "Path": "/"
            },
            "LastAuthenticated": "2019-09-30T23:02:00Z"
        },
        {
            "EntityInfo": {
                "Arn": "arn:aws:iam::123456789012:user/developer",
                "Name": "developer",
```

```
                "Type": "USER",
                "Id": "AIDAIBEYXMPL2YEXAMPLE",
                "Path": "/"
            },
            "LastAuthenticated": "2019-09-16T19:34:00Z"
        }
    ]
}
```

For more information, see Refining permissions in AWS using last accessed information in the *AWS IAM User Guide*.

- For API details, see GetServiceLastAccessedDetailsWithEntities in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example provides the last accessed timestamp for the service in the request by that respective IAM entity.

```
$results = Get-IAMServiceLastAccessedDetailWithEntity -JobId f0b7a819-eab0-929b-
dc26-ca598911cb9f -ServiceNamespace ec2
$results
```

Output:

```
EntityDetailsList : {Amazon.IdentityManagement.Model.EntityDetails}
Error             :
IsTruncated       : False
JobCompletionDate : 12/29/19 11:19:31 AM
JobCreationDate   : 12/29/19 11:19:31 AM
JobStatus         : COMPLETED
Marker            :
```

```
$results.EntityDetailsList
```

Output:

```
EntityInfo                                  LastAuthenticated
```

```
----------                                      -----------------
Amazon.IdentityManagement.Model.EntityInfo 11/16/19 3:47:00 PM
```

```
$results.EntityInfo
```

Output:

```
Arn  : arn:aws:iam::123456789012:user/TestUser
Id   : AIDA4NBK5CXF5TZHU1234
Name : TestUser
Path : /
Type : USER
```

- For API details, see GetServiceLastAccessedDetailsWithEntities in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example provides the last accessed timestamp for the service in the request by that respective IAM entity.

```
$results = Get-IAMServiceLastAccessedDetailWithEntity -JobId f0b7a819-eab0-929b-
dc26-ca598911cb9f -ServiceNamespace ec2
$results
```

Output:

```
EntityDetailsList : {Amazon.IdentityManagement.Model.EntityDetails}
Error             :
IsTruncated       : False
JobCompletionDate : 12/29/19 11:19:31 AM
JobCreationDate   : 12/29/19 11:19:31 AM
JobStatus         : COMPLETED
Marker            :
```

```
$results.EntityDetailsList
```

Output:

```
EntityInfo                                  LastAuthenticated
```

```
----------                                      -----------------
Amazon.IdentityManagement.Model.EntityInfo 11/16/19 3:47:00 PM
```

```
$results.EntityInfo
```

Output:

```
Arn  : arn:aws:iam::123456789012:user/TestUser
Id   : AIDA4NBK5CXF5TZHU1234
Name : TestUser
Path : /
Type : USER
```

- For API details, see GetServiceLastAccessedDetailsWithEntities in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetServiceLinkedRoleDeletionStatus` with an AWS SDK or CLI

The following code examples show how to use `GetServiceLinkedRoleDeletionStatus`.

CLI

AWS CLI

To check the status of a request to delete a service-linked role

The following `get-service-linked-role-deletion-status` example displays the status of a previously request to delete a service-linked role. The delete operation occurs asynchronously. When you make the request, you get a `DeletionTaskId` value that you provide as a parameter for this command.

```
aws iam get-service-linked-role-deletion-status \
    --deletion-task-id task/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

Output:

```
{
"Status": "SUCCEEDED"
}
```

For more information, see Using service-linked roles in the *AWS IAM User Guide*.

- For API details, see GetServiceLinkedRoleDeletionStatus in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- For API details, see GetServiceLinkedRoleDeletionStatus in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetUser` with an AWS SDK or CLI

The following code examples show how to use `GetUser`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest
{ UserName = userName });
        return response.User;
    }
```

- For API details, see GetUser in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function iam_user_exists
#
# This function checks to see if the specified AWS Identity and Access Management
 (IAM) user already exists.
#
# Parameters:
#       $1 - The name of the IAM user to check.
#
# Returns:
#       0 - If the user already exists.
#       1 - If the user doesn't exist.
##############################################################################
function iam_user_exists() {
  local user_name
  user_name=$1

  # Check whether the IAM user already exists.
  # We suppress all output - we're interested only in the return code.

  local errors
  errors=$(aws iam get-user \
    --user-name "$user_name" 2>&1 >/dev/null)
```

```
  local error_code=${?}

  if [[ $error_code -eq 0 ]]; then
    return 0 # 0 in Bash script means true.
  else
    if [[ $errors != *"error"*"(NoSuchEntity)"* ]]; then
      aws_cli_error_log $error_code
      errecho "Error calling iam get-user $errors"
    fi

    return 1 # 1 in Bash script means false.
  fi
}
```

- For API details, see GetUser in *AWS CLI Command Reference*.

CLI

AWS CLI

To get information about an IAM user

The following `get-user` example gets information about the IAM user named `Paulo`.

```
aws iam get-user \
    --user-name Paulo
```

Output:

```
{
    "User": {
        "UserName": "Paulo",
        "Path": "/",
        "CreateDate": "2019-09-21T23:03:13Z",
        "UserId": "AIDA123456789EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:user/Paulo"
    }
}
```

For more information, see Managing IAM users in the *AWS IAM User Guide*.

- For API details, see GetUser in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.GetUser(ctx, &iam.GetUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  var apiError smithy.APIError
```

```
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NoSuchEntityException:
    log.Printf("User %v does not exist.\n", userName)
    err = nil
   default:
    log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
   }
  }
 } else {
  user = result.User
 }
 return user, err
}
```

- For API details, see GetUser in *AWS SDK for Go API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves details about the user named `David`.

```
Get-IAMUser -UserName David
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/David
CreateDate       : 12/10/2014 3:39:27 PM
PasswordLastUsed : 3/19/2015 8:44:04 AM
Path             : /
UserId           : Y4FKWQCXTA52QEXAMPLE1
UserName         : David
```

Example 2: This example retrieves details about the currently signed-in IAM user.

```
Get-IAMUser
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Bob
CreateDate       : 10/16/2014 9:03:09 AM
PasswordLastUsed : 3/4/2015 12:12:33 PM
Path             : /
UserId           : 7K3GJEANSKZF2EXAMPLE2
UserName         : Bob
```

- For API details, see GetUser in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves details about the user named `David`.

```
Get-IAMUser -UserName David
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/David
CreateDate       : 12/10/2014 3:39:27 PM
PasswordLastUsed : 3/19/2015 8:44:04 AM
Path             : /
UserId           : Y4FKWQCXTA52QEXAMPLE1
UserName         : David
```

Example 2: This example retrieves details about the currently signed-in IAM user.

```
Get-IAMUser
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Bob
CreateDate       : 10/16/2014 9:03:09 AM
PasswordLastUsed : 3/4/2015 12:12:33 PM
Path             : /
UserId           : 7K3GJEANSKZF2EXAMPLE2
UserName         : Bob
```

- For API details, see GetUser in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
 # Retrieves a user's details
 #
 # @param user_name [String] The name of the user to retrieve
 # @return [Aws::IAM::Types::User, nil] The user object if found, or nil if an
error occurred
 def get_user(user_name)
   response = @iam_client.get_user(user_name: user_name)
   response.user
 rescue Aws::IAM::Errors::NoSuchEntity
   @logger.error("User '#{user_name}' not found.")
   nil
 rescue Aws::IAM::Errors::ServiceError => e
   @logger.error("Error retrieving user '#{user_name}': #{e.message}")
   nil
 end
```

- For API details, see GetUser in *AWS SDK for Ruby API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
```

```
import AWSS3


    public func getUser(name: String? = nil) async throws -> IAMClientTypes.User
 {
        let input = GetUserInput(
            userName: name
        )
        do {
            let output = try await iamClient.getUser(input: input)
            guard let user = output.user else {
                throw ServiceHandlerError.noSuchUser
            }
            return user
        } catch {
            print("ERROR: getUser:", dump(error))
            throw error
        }
    }
```

- For API details, see GetUser in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetUserPolicy` with a CLI

The following code examples show how to use `GetUserPolicy`.

CLI

AWS CLI

To list policy details for an IAM user

The following `get-user-policy` example lists the details of the specified policy that is attached to the IAM user named Bob.

```
aws iam get-user-policy \
    --user-name Bob \
```

```
    --policy-name ExamplePolicy
```

Output:

```
{
    "UserName": "Bob",
    "PolicyName": "ExamplePolicy",
    "PolicyDocument": {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Action": "*",
                "Resource": "*",
                "Effect": "Allow"
            }
        ]
    }
}
```

To get a list of policies for an IAM user, use the `list-user-policies` command.

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see GetUserPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves the details of the inline policy named `Davids_IAM_Admin_Policy` that is embedded in the IAM user named `David`. The policy document is URL encoded.

```
$results = Get-IAMUserPolicy -PolicyName Davids_IAM_Admin_Policy -UserName David
$results
```

Output:

```
PolicyDocument                                          PolicyName
       UserName
--------------                                          ----------
       --------
```

```
%7B%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C%...    Davids_IAM_Admin_Policy
       David

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
[System.Web.HttpUtility]::UrlDecode($results.PolicyDocument)
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetUser",
        "iam:ListUsers"
      ],
      "Resource": [
        "arn:aws:iam::111122223333:user/*"
      ]
    }
  ]
}
```

- For API details, see GetUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves the details of the inline policy named `Davids_IAM_Admin_Policy` that is embedded in the IAM user named `David`. The policy document is URL encoded.

```
$results = Get-IAMUserPolicy -PolicyName Davids_IAM_Admin_Policy -UserName David
$results
```

Output:

```
PolicyDocument                                           PolicyName
       UserName
--------------                                           ----------
       --------
%7B%0A%20%20%22Version%22%3A%20%222012-10-17%22%2C%...    Davids_IAM_Admin_Policy
       David

[System.Reflection.Assembly]::LoadWithPartialName("System.Web.HttpUtility")
[System.Web.HttpUtility]::UrlDecode($results.PolicyDocument)
```

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetUser",
        "iam:ListUsers"
      ],
      "Resource": [
        "arn:aws:iam::111122223333:user/*"
      ]
    }
  ]
}
```

- For API details, see GetUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListAccessKeys` with an AWS SDK or CLI

The following code examples show how to use `ListAccessKeys`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage access keys

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function iam_list_access_keys
#
# This function lists the access keys for the specified user.
#
# Parameters:
#       -u user_name -- The name of the IAM user.
#
# Returns:
#       access_key_ids
#     And:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function iam_list_access_keys() {

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_list_access_keys"
    echo "Lists the AWS Identity and Access Management (IAM) access key IDs for
 the specified user."
    echo "  -u user_name   The name of the IAM user."
    echo ""
  }

  local user_name response
  local option OPTARG # Required to use getopts command in a function.
  # Retrieve the calling parameters.
  while getopts "u:h" option; do
    case "${option}" in
      u) user_name="${OPTARG}" ;;
      h)
        usage
        return 0
```

```
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$user_name" ]]; then
    errecho "ERROR: You must provide a username with the -u parameter."
    usage
    return 1
  fi

  response=$(aws iam list-access-keys \
    --user-name "$user_name" \
    --output text \
    --query 'AccessKeyMetadata[].AccessKeyId')

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-access-keys operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- For API details, see ListAccessKeys in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::listAccessKeys(const Aws::String &userName,
                                 const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListAccessKeysRequest request;
    request.SetUserName(userName);

    bool done = false;
    bool header = false;
    while (!done) {
        auto outcome = iam.ListAccessKeys(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list access keys for user " << userName
                      << ": " << outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        if (!header) {
            std::cout << std::left << std::setw(32) << "UserName" <<
                      std::setw(30) << "KeyID" << std::setw(20) << "Status" <<
                      std::setw(20) << "CreateDate" << std::endl;
            header = true;
        }

        const auto &keys = outcome.GetResult().GetAccessKeyMetadata();
        const Aws::String DATE_FORMAT = "%Y-%m-%d";

        for (const auto &key: keys) {
            Aws::String statusString =
                    Aws::IAM::Model::StatusTypeMapper::GetNameForStatusType(
                            key.GetStatus());
            std::cout << std::left << std::setw(32) << key.GetUserName() <<
```

```
                        std::setw(30) << key.GetAccessKeyId() << std::setw(20) <<
                        statusString << std::setw(20) <<
                        key.GetCreateDate().ToGmtString(DATE_FORMAT.c_str()) <<
 std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }

    return true;
}
```

- For API details, see ListAccessKeys in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list the access key IDs for an IAM user

The following `list-access-keys` example lists the access keys IDs for the IAM user named Bob.

```
aws iam list-access-keys \
    --user-name Bob
```

Output:

```
{
    "AccessKeyMetadata": [
        {
            "UserName": "Bob",
            "Status": "Active",
            "CreateDate": "2013-06-04T18:17:34Z",
            "AccessKeyId": "AKIAIOSFODNN7EXAMPLE"
        },
```

```
        {
            "UserName": "Bob",
            "Status": "Inactive",
            "CreateDate": "2013-06-06T20:42:26Z",
            "AccessKeyId": "AKIAI44QH8DHBEXAMPLE"
        }
    ]
}
```

You cannot list the secret access keys for IAM users. If the secret access keys are lost, you must create new access keys using the `create-access-keys` command.

For more information, see Managing access keys for IAM users in the *AWS IAM User Guide*.

- For API details, see ListAccessKeys in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
```

```
 IamClient *iam.Client
}



// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(ctx context.Context, userName string)
 ([]types.AccessKeyMetadata, error) {
 var keys []types.AccessKeyMetadata
 result, err := wrapper.IamClient.ListAccessKeys(ctx, &iam.ListAccessKeysInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
 err)
 } else {
  keys = result.AccessKeyMetadata
 }
 return keys, err
}
```

- For API details, see ListAccessKeys in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.AccessKeyMetadata;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.ListAccessKeysRequest;
import software.amazon.awssdk.services.iam.model.ListAccessKeysResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListAccessKeys {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <userName>\s

                Where:
                    userName - The name of the user for which access keys are
 retrieved.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String userName = args[0];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        listKeys(iam, userName);
        System.out.println("Done");
        iam.close();
    }

    public static void listKeys(IamClient iam, String userName) {
        try {
            boolean done = false;
            String newMarker = null;

            while (!done) {
                ListAccessKeysResponse response;
```

```
                if (newMarker == null) {
                    ListAccessKeysRequest request =
 ListAccessKeysRequest.builder()
                            .userName(userName)
                            .build();

                    response = iam.listAccessKeys(request);

                } else {
                    ListAccessKeysRequest request =
 ListAccessKeysRequest.builder()
                            .userName(userName)
                            .marker(newMarker)
                            .build();

                    response = iam.listAccessKeys(request);
                }

                for (AccessKeyMetadata metadata : response.accessKeyMetadata()) {
                    System.out.format("Retrieved access key %s",
 metadata.accessKeyId());
                }

                if (!response.isTruncated()) {
                    done = true;
                } else {
                    newMarker = response.marker();
                }
            }

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListAccessKeys in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the access keys.

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput |
 undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
    for (const key of response.AccessKeyMetadata) {
      yield key;
    }

    if (response.IsTruncated) {
      response = await client.send(
```

```
        new ListAccessKeysCommand({
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListAccessKeys in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

iam.listAccessKeys(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListAccessKeys in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listKeys(userNameVal: String?) {
    val request =
        ListAccessKeysRequest {
            userName = userNameVal
        }
    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAccessKeys(request)
        response.accessKeyMetadata?.forEach { md ->
            println("Retrieved access key ${md.accessKeyId}")
        }
    }
}
```

- For API details, see ListAccessKeys in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command lists the access keys for the IAM user named Bob. Note that you cannot list the secret access keys for IAM users. If the secret access keys are lost, you must create new access keys with the `New-IAMAccessKey` cmdlet.

```
Get-IAMAccessKey -UserName "Bob"
```

Output:

```
AccessKeyId               CreateDate                 Status
 UserName
-----------               ----------                 ------
 --------
AKIAIOSFODNN7EXAMPLE      12/3/2014 10:53:41 AM      Active             Bob
AKIAI44QH8DHBEXAMPLE      6/6/2013 8:42:26 PM        Inactive           Bob
```

- For API details, see ListAccessKeys in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command lists the access keys for the IAM user named Bob. Note that you cannot list the secret access keys for IAM users. If the secret access keys are lost, you must create new access keys with the `New-IAMAccessKey` cmdlet.

```
Get-IAMAccessKey -UserName "Bob"
```

Output:

```
AccessKeyId               CreateDate                 Status
 UserName
-----------               ----------                 ------
 --------
AKIAIOSFODNN7EXAMPLE      12/3/2014 10:53:41 AM      Active             Bob
AKIAI44QH8DHBEXAMPLE      6/6/2013 8:42:26 PM        Inactive           Bob
```

- For API details, see ListAccessKeys in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_keys(user_name):
    """
    Lists the keys owned by the specified user.

    :param user_name: The name of the user.
    :return: The list of keys owned by the user.
    """
    try:
        keys = list(iam.User(user_name).access_keys.all())
        logger.info("Got %s access keys for %s.", len(keys), user_name)
    except ClientError:
        logger.exception("Couldn't get access keys for %s.", user_name)
        raise
    else:
        return keys
```

- For API details, see ListAccessKeys in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, deactivates, and deletes access keys.

```
# Manages access keys for IAM users
class AccessKeyManager
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'AccessKeyManager'
  end
```

```
  # Lists access keys for a user
  #
  # @param user_name [String] The name of the user.
  def list_access_keys(user_name)
    response = @iam_client.list_access_keys(user_name: user_name)
    if response.access_key_metadata.empty?
      @logger.info("No access keys found for user '#{user_name}'.")
    else
      response.access_key_metadata.map(&:access_key_id)
    end
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Error listing access keys: cannot find user '#{user_name}'.")
    []
  rescue StandardError => e
    @logger.error("Error listing access keys: #{e.message}")
    []
  end

  # Creates an access key for a user
  #
  # @param user_name [String] The name of the user.
  # @return [Boolean]
  def create_access_key(user_name)
    response = @iam_client.create_access_key(user_name: user_name)
    access_key = response.access_key
    @logger.info("Access key created for user '#{user_name}':
#{access_key.access_key_id}")
    access_key
  rescue Aws::IAM::Errors::LimitExceeded
    @logger.error('Error creating access key: limit exceeded. Cannot create
more.')
    nil
  rescue StandardError => e
    @logger.error("Error creating access key: #{e.message}")
    nil
  end

  # Deactivates an access key
  #
  # @param user_name [String] The name of the user.
  # @param access_key_id [String] The ID for the access key.
  # @return [Boolean]
  def deactivate_access_key(user_name, access_key_id)
    @iam_client.update_access_key(
```

```
      user_name: user_name,
      access_key_id: access_key_id,
      status: 'Inactive'
    )
    true
  rescue StandardError => e
    @logger.error("Error deactivating access key: #{e.message}")
    false
  end

  # Deletes an access key
  #
  # @param user_name [String] The name of the user.
  # @param access_key_id [String] The ID for the access key.
  # @return [Boolean]
  def delete_access_key(user_name, access_key_id)
    @iam_client.delete_access_key(
      user_name: user_name,
      access_key_id: access_key_id
    )
    true
  rescue StandardError => e
    @logger.error("Error deleting access key: #{e.message}")
    false
  end
end
```

- For API details, see ListAccessKeys in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listaccesskeys(
```

```
          iv_username = iv_user_name ).
        MESSAGE 'Retrieved access key list.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'User does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListAccessKeys in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListAccountAliases` with an AWS SDK or CLI

The following code examples show how to use `ListAccountAliases`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage your account

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool
AwsDoc::IAM::listAccountAliases(const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListAccountAliasesRequest request;

    bool done = false;
    bool header = false;
```

```
    while (!done) {
        auto outcome = iam.ListAccountAliases(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list account aliases: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        const auto &aliases = outcome.GetResult().GetAccountAliases();
        if (!header) {
            if (aliases.size() == 0) {
                std::cout << "Account has no aliases" << std::endl;
                break;
            }
            std::cout << std::left << std::setw(32) << "Alias" << std::endl;
            header = true;
        }

        for (const auto &alias: aliases) {
            std::cout << std::left << std::setw(32) << alias << std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }

    return true;
}
```

- For API details, see ListAccountAliases in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list account aliases

The following `list-account-aliases` example lists the aliases for the current account.

```
aws iam list-account-aliases
```

Output:

```
{
    "AccountAliases": [
    "mycompany"
    ]
}
```

For more information, see Your AWS account ID and its alias in the *AWS IAM User Guide*.

- For API details, see ListAccountAliases in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.ListAccountAliasesResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListAccountAliases {
    public static void main(String[] args) {
        Region region = Region.AWS_GLOBAL;
```

```
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        listAliases(iam);
        System.out.println("Done");
        iam.close();
    }

    public static void listAliases(IamClient iam) {
        try {
            ListAccountAliasesResponse response = iam.listAccountAliases();
            for (String alias : response.accountAliases()) {
                System.out.printf("Retrieved account alias %s", alias);
            }

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListAccountAliases in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the account aliases.

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
 */
export async function* listAccountAliases() {
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });

  let response = await client.send(command);

  while (response.AccountAliases?.length) {
    for (const alias of response.AccountAliases) {
      yield alias;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccountAliasesCommand({
          Marker: response.Marker,
          MaxItems: 5,
        }),
      );
    } else {
      break;
    }
  }
}
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListAccountAliases in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListAccountAliases in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listAliases() {
    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listAccountAliases(ListAccountAliasesRequest {})
        response.accountAliases?.forEach { alias ->
            println("Retrieved account alias $alias")
        }
    }
}
```

- For API details, see ListAccountAliases in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command returns the account alias for the AWS account.

```
Get-IAMAccountAlias
```

Output:

```
ExampleCo
```

- For API details, see ListAccountAliases in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command returns the account alias for the AWS account.

```
Get-IAMAccountAlias
```

Output:

```
ExampleCo
```

- For API details, see ListAccountAliases in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_aliases():
    """
```

```
    Gets the list of aliases for the current account. An account has at most one
 alias.

    :return: The list of aliases for the account.
    """
    try:
        response = iam.meta.client.list_account_aliases()
        aliases = response["AccountAliases"]
        if len(aliases) > 0:
            logger.info("Got aliases for your account: %s.", ",".join(aliases))
        else:
            logger.info("Got no aliases for your account.")
    except ClientError:
        logger.exception("Couldn't list aliases for your account.")
        raise
    else:
        return response["AccountAliases"]
```

- For API details, see ListAccountAliases in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List, create, and delete account aliases.

```
class IAMAliasManager
  # Initializes the IAM client and logger
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client.
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
```

```
  end

  # Lists available AWS account aliases.
  def list_aliases
    response = @iam_client.list_account_aliases

    if response.account_aliases.count.positive?
      @logger.info('Account aliases are:')
      response.account_aliases.each { |account_alias| @logger.info("
 #{account_alias}") }
    else
      @logger.info('No account aliases found.')
    end
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing account aliases: #{e.message}")
  end

  # Creates an AWS account alias.
  #
  # @param account_alias [String] The name of the account alias to create.
  # @return [Boolean] true if the account alias was created; otherwise, false.
  def create_account_alias(account_alias)
    @iam_client.create_account_alias(account_alias: account_alias)
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating account alias: #{e.message}")
    false
  end

  # Deletes an AWS account alias.
  #
  # @param account_alias [String] The name of the account alias to delete.
  # @return [Boolean] true if the account alias was deleted; otherwise, false.
  def delete_account_alias(account_alias)
    @iam_client.delete_account_alias(account_alias: account_alias)
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting account alias: #{e.message}")
    false
  end
end
```

- For API details, see ListAccountAliases in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->listaccountaliases( ).
    MESSAGE 'Retrieved account alias list.' TYPE 'I'.
  CATCH /aws1/cx_iamservicefailureex.
    MESSAGE 'Service failure when listing account aliases.' TYPE 'E'.
ENDTRY.
```

- For API details, see ListAccountAliases in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListAttachedGroupPolicies` with a CLI

The following code examples show how to use `ListAttachedGroupPolicies`.

CLI

AWS CLI

To list all managed policies that are attached to the specified group

The following `list-attached-group-policies` example returns the names and ARNs of the managed policies that are attached to the IAM group named `Admins` in the AWS account.

```
aws iam list-attached-group-policies \
    --group-name Admins
```

Output:

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AdministratorAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
        },
        {
            "PolicyName": "SecurityAudit",
            "PolicyArn": "arn:aws:iam::aws:policy/SecurityAudit"
        }
    ],
    "IsTruncated": false
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see ListAttachedGroupPolicies in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command returns the names and ARNs of the managed policies that are attached to the IAM group named `Admins` in the AWS account. To see the list of inline policies embedded in the group, use the `Get-IAMGroupPolicyList` command.

```
Get-IAMAttachedGroupPolicyList -GroupName "Admins"
```

Output:

```
PolicyArn                                                 PolicyName
---------                                                 ----------
arn:aws:iam::aws:policy/SecurityAudit                     SecurityAudit
arn:aws:iam::aws:policy/AdministratorAccess               AdministratorAccess
```

- For API details, see ListAttachedGroupPolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command returns the names and ARNs of the managed policies that are attached to the IAM group named `Admins` in the AWS account. To see the list of inline policies embedded in the group, use the `Get-IAMGroupPolicyList` command.

```
Get-IAMAttachedGroupPolicyList -GroupName "Admins"
```

Output:

```
PolicyArn                                                 PolicyName
---------                                                 ----------
arn:aws:iam::aws:policy/SecurityAudit                     SecurityAudit
arn:aws:iam::aws:policy/AdministratorAccess               AdministratorAccess
```

- For API details, see ListAttachedGroupPolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListAttachedRolePolicies` with an AWS SDK or CLI

The following code examples show how to use `ListAttachedRolePolicies`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
```

```
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>>
ListAttachedRolePoliciesAsync(string roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new
ListAttachedRolePoliciesRequest { RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list all managed policies that are attached to the specified role

The following `list-attached-role-policies` example returns the names and ARNs of the managed policies attached to the IAM role named `SecurityAuditRole` in the AWS account.

```
aws iam list-attached-role-policies \
    --role-name SecurityAuditRole
```

Output:

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "SecurityAudit",
            "PolicyArn": "arn:aws:iam::aws:policy/SecurityAudit"
        }
```

```
    ],
    "IsTruncated": false
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see ListAttachedRolePolicies in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListAttachedRolePolicies lists the policies that are attached to the specified
 role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(ctx context.Context, roleName
 string) ([]types.AttachedPolicy, error) {
```

```
 var policies []types.AttachedPolicy
 result, err := wrapper.IamClient.ListAttachedRolePolicies(ctx,
 &iam.ListAttachedRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
 roleName, err)
 } else {
  policies = result.AttachedPolicies
 }
 return policies, err
}
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the policies that are attached to a role.

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
```

```
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);

  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
          RoleName: roleName,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();
```

```
    public function listAttachedRolePolicies($roleName, $pathPrefix = "", $marker
 = "", $maxItems = 0)
    {
        $listAttachRolePoliciesArguments = ['RoleName' => $roleName];
        if ($pathPrefix) {
            $listAttachRolePoliciesArguments['PathPrefix'] = $pathPrefix;
        }
        if ($marker) {
            $listAttachRolePoliciesArguments['Marker'] = $marker;
        }
        if ($maxItems) {
            $listAttachRolePoliciesArguments['MaxItems'] = $maxItems;
        }
        return $this->iamClient-
>listAttachedRolePolicies($listAttachRolePoliciesArguments);
    }
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command returns the names and ARNs of the managed policies attached to the IAM role named `SecurityAuditRole` in the AWS account. To see the list of inline policies that are embedded in the role, use the `Get-IAMRolePolicyList` command.

```
Get-IAMAttachedRolePolicyList -RoleName "SecurityAuditRole"
```

Output:

```
PolicyArn                                                 PolicyName
---------                                                 ----------
arn:aws:iam::aws:policy/SecurityAudit                     SecurityAudit
```

- For API details, see ListAttachedRolePolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command returns the names and ARNs of the managed policies attached to the IAM role named `SecurityAuditRole` in the AWS account. To see the list of inline policies that are embedded in the role, use the `Get-IAMRolePolicyList` command.

```
Get-IAMAttachedRolePolicyList -RoleName "SecurityAuditRole"
```

Output:

```
PolicyArn                                                 PolicyName
---------                                                 ----------
arn:aws:iam::aws:policy/SecurityAudit                     SecurityAudit
```

- For API details, see ListAttachedRolePolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_attached_policies(role_name):
    """
    Lists policies attached to a role.

    :param role_name: The name of the role to query.
    """
    try:
        role = iam.Role(role_name)
        for policy in role.attached_policies.all():
            logger.info("Got policy %s.", policy.arn)
    except ClientError:
        logger.exception("Couldn't list attached policies for %s.", role_name)
        raise
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, attaches, and detaches role policies.

```
# Manages policies in AWS Identity and Access Management (IAM)
class RolePolicyManager
  # Initialize with an AWS IAM client
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'PolicyManager'
  end

  # Creates a policy
  #
  # @param policy_name [String] The name of the policy
  # @param policy_document [Hash] The policy document
  # @return [String] The policy ARN if successful, otherwise nil
  def create_policy(policy_name, policy_document)
    response = @iam_client.create_policy(
      policy_name: policy_name,
      policy_document: policy_document.to_json
    )
    response.policy.arn
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating policy: #{e.message}")
```

```
    nil
  end

  # Fetches an IAM policy by its ARN
  # @param policy_arn [String] the ARN of the IAM policy to retrieve
  # @return [Aws::IAM::Types::GetPolicyResponse] the policy object if found
  def get_policy(policy_arn)
    response = @iam_client.get_policy(policy_arn: policy_arn)
    policy = response.policy
    @logger.info("Got policy '#{policy.policy_name}'. Its ID is:
#{policy.policy_id}.")
    policy
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Couldn't get policy '#{policy_arn}'. The policy does not
exist.")
    raise
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't get policy '#{policy_arn}'. Here's why: #{e.code}:
#{e.message}")
    raise
  end

  # Attaches a policy to a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def attach_policy_to_role(role_name, policy_arn)
    @iam_client.attach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error attaching policy to role: #{e.message}")
    false
  end

  # Lists policy ARNs attached to a role
  #
  # @param role_name [String] The name of the role
  # @return [Array<String>] List of policy ARNs
  def list_attached_policy_arns(role_name)
    response = @iam_client.list_attached_role_policies(role_name: role_name)
```

```
    response.attached_policies.map(&:policy_arn)
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing policies attached to role: #{e.message}")
    []
  end

  # Detaches a policy from a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def detach_policy_from_role(role_name, policy_arn)
    @iam_client.detach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error detaching policy from role: #{e.message}")
    false
  end
end
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn list_attached_role_policies(
    client: &iamClient,
    role_name: String,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
```

```
) -> Result<ListAttachedRolePoliciesOutput,
 SdkError<ListAttachedRolePoliciesError>> {
    let response = client
        .list_attached_role_policies()
        .role_name(role_name)
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listattachedrolepolicies(
          iv_rolename = iv_role_name ).
        MESSAGE 'Retrieved attached policy list for role.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Role does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3



    /// Returns a list of AWS Identity and Access Management (IAM) policies
    /// that are attached to the role.
    ///
    /// - Parameter role: The IAM role to return the policy list for.
    ///
    /// - Returns: An array of `IAMClientTypes.AttachedPolicy` objects
    ///   describing each managed policy that's attached to the role.
    public func listAttachedRolePolicies(role: String) async throws ->
 [IAMClientTypes.AttachedPolicy] {
        var policyList: [IAMClientTypes.AttachedPolicy] = []

        // Use "Paginated" to get all the attached role polices.
        // This lets the SDK handle the 'isTruncated' in
 "ListAttachedRolePoliciesOutput".
        let input = ListAttachedRolePoliciesInput(
            roleName: role
        )
        let output = client.listAttachedRolePoliciesPaginated(input: input)

        do {
            for try await page in output {
                guard let attachedPolicies = page.attachedPolicies else {
                    print("Error: no attached policies returned.")
                    continue
                }
                for attachedPolicy in attachedPolicies {
                    policyList.append(attachedPolicy)
```

```
                }
            }
        } catch {
            print("ERROR: listAttachedRolePolicies:", dump(error))
            throw error
        }

        return policyList
    }
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListAttachedUserPolicies` with a CLI

The following code examples show how to use `ListAttachedUserPolicies`.

CLI

AWS CLI

To list all managed policies that are attached to the specified user

The following `list-attached-user-policies` example returns the names and ARNs of the managed policies for the IAM user named Bob in the AWS account.

```
aws iam list-attached-user-policies \
    --user-name Bob
```

Output:

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AdministratorAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
        },
        {
```

```
            "PolicyName": "SecurityAudit",
            "PolicyArn": "arn:aws:iam::aws:policy/SecurityAudit"
        }
    ],
    "IsTruncated": false
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see ListAttachedUserPolicies in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This command returns the names and ARNs of the managed policies for the IAM user named Bob in the AWS account. To see the list of inline policies that are embedded in the IAM user, use the `Get-IAMUserPolicyList` command.

```
Get-IAMAttachedUserPolicyList -UserName "Bob"
```

Output:

```
PolicyArn                                                 PolicyName
---------                                                 ----------
arn:aws:iam::aws:policy/TesterPolicy                      TesterPolicy
```

- For API details, see ListAttachedUserPolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This command returns the names and ARNs of the managed policies for the IAM user named Bob in the AWS account. To see the list of inline policies that are embedded in the IAM user, use the `Get-IAMUserPolicyList` command.

```
Get-IAMAttachedUserPolicyList -UserName "Bob"
```

Output:

```
PolicyArn                                                 PolicyName
```

```
---------                                                    ----------
arn:aws:iam::aws:policy/TesterPolicy                         TesterPolicy
```

- For API details, see ListAttachedUserPolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListEntitiesForPolicy` with a CLI

The following code examples show how to use `ListEntitiesForPolicy`.

CLI

AWS CLI

To list all users, groups, and roles that the specified managed policy is attached to

The following `list-entities-for-policy` example returns a list of IAM groups, roles, and users who have the policy `arn:aws:iam::123456789012:policy/TestPolicy` attached.

```
aws iam list-entities-for-policy \
    --policy-arn arn:aws:iam::123456789012:policy/TestPolicy
```

Output:

```
{
    "PolicyGroups": [
        {
            "GroupName": "Admins",
            "GroupId": "AGPACKCEVSQ6C2EXAMPLE"
        }
    ],
    "PolicyUsers": [
        {
            "UserName": "Alice",
            "UserId": "AIDACKCEVSQ6C2EXAMPLE"
        }
```

```
    ],
    "PolicyRoles": [
        {
            "RoleName": "DevRole",
            "RoleId": "AROADBQP57FF2AEXAMPLE"
        }
    ],
    "IsTruncated": false
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see ListEntitiesForPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a list of IAM groups, roles, and users who have the policy `arn:aws:iam::123456789012:policy/TestPolicy` attached.

```
Get-IAMEntitiesForPolicy -PolicyArn "arn:aws:iam::123456789012:policy/TestPolicy"
```

Output:

```
IsTruncated  : False
Marker       :
PolicyGroups : {}
PolicyRoles  : {testRole}
PolicyUsers  : {Bob, Theresa}
```

- For API details, see ListEntitiesForPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a list of IAM groups, roles, and users who have the policy `arn:aws:iam::123456789012:policy/TestPolicy` attached.

```
Get-IAMEntitiesForPolicy -PolicyArn "arn:aws:iam::123456789012:policy/TestPolicy"
```

Output:

```
IsTruncated  : False
Marker       :
PolicyGroups : {}
PolicyRoles  : {testRole}
PolicyUsers  : {Bob, Theresa}
```

- For API details, see ListEntitiesForPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListGroupPolicies` with a CLI

The following code examples show how to use `ListGroupPolicies`.

CLI

AWS CLI

To list all inline policies that are attached to the specified group

The following `list-group-policies` example lists the names of inline policies that are attached to the IAM group named `Admins` in the current account.

```
aws iam list-group-policies \
    --group-name Admins
```

Output:

```
{
    "PolicyNames": [
        "AdminRoot",
        "ExamplePolicy"
    ]
}
```

For more information, see Managing IAM policies in the *AWS IAM User Guide*.

- For API details, see ListGroupPolicies in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a list of the inline policies that are embedded in the group `Testers`. To get the managed policies that are attached to the group, use the command `Get-IAMAttachedGroupPolicyList`.

```
Get-IAMGroupPolicyList -GroupName Testers
```

Output:

```
Deny-Assume-S3-Role-In-Production
PowerUserAccess-Testers
```

- For API details, see ListGroupPolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a list of the inline policies that are embedded in the group `Testers`. To get the managed policies that are attached to the group, use the command `Get-IAMAttachedGroupPolicyList`.

```
Get-IAMGroupPolicyList -GroupName Testers
```

Output:

```
Deny-Assume-S3-Role-In-Production
PowerUserAccess-Testers
```

- For API details, see ListGroupPolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListGroups` with an AWS SDK or CLI

The following code examples show how to use `ListGroups`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List IAM groups.
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }
```

- For API details, see ListGroups in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list the IAM groups for the current account

The following `list-groups` example lists the IAM groups in the current account.

```
aws iam list-groups
```

Output:

```
{
    "Groups": [
        {
            "Path": "/",
            "CreateDate": "2013-06-04T20:27:27.972Z",
            "GroupId": "AIDACKCEVSQ6C2EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:group/Admins",
            "GroupName": "Admins"
        },
        {
            "Path": "/",
            "CreateDate": "2013-04-16T20:30:42Z",
            "GroupId": "AIDGPMS9RO4H3FEXAMPLE",
            "Arn": "arn:aws:iam::123456789012:group/S3-Admins",
            "GroupName": "S3-Admins"
        }
    ]
}
```

For more information, see Managing IAM user groups in the *AWS IAM User Guide*.

- For API details, see ListGroups in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// GroupWrapper encapsulates AWS Identity and Access Management (IAM) group
 actions
// used in the examples.
// It contains an IAM service client that is used to perform group actions.
type GroupWrapper struct {
 IamClient *iam.Client
}



// ListGroups lists up to maxGroups number of groups.
func (wrapper GroupWrapper) ListGroups(ctx context.Context, maxGroups int32)
 ([]types.Group, error) {
 var groups []types.Group
 result, err := wrapper.IamClient.ListGroups(ctx, &iam.ListGroupsInput{
  MaxItems: aws.Int32(maxGroups),
 })
 if err != nil {
  log.Printf("Couldn't list groups. Here's why: %v\n", err)
 } else {
  groups = result.Groups
 }
 return groups, err
}
```

- For API details, see ListGroups in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the groups.

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
 */
export async function* listGroups() {
  const command = new ListGroupsCommand({
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.Groups?.length) {
    for (const group of response.Groups) {
      yield group;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListGroupsCommand({
          Marker: response.Marker,
          MaxItems: 10,
        }),
      );
    } else {
      break;
    }
  }
}
```

- For API details, see ListGroups in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listGroups($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listGroupsArguments = [];
        if ($pathPrefix) {
            $listGroupsArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listGroupsArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listGroupsArguments["MaxItems"] = $maxItems;
        }

        return $this->iamClient->listGroups($listGroupsArguments);
    }
```

- For API details, see ListGroups in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a collection of all the IAM groups defined in the current AWS account.

```
Get-IAMGroupList
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Administrators
CreateDate : 10/20/2014 10:06:24 AM
GroupId    : 6WCH4TRY3KIHIEXAMPLE1
GroupName  : Administrators
Path       : /

Arn        : arn:aws:iam::123456789012:group/Developers
CreateDate : 12/10/2014 3:38:55 PM
GroupId    : ZU2EOWMK6WBZOEXAMPLE2
GroupName  : Developers
Path       : /

Arn        : arn:aws:iam::123456789012:group/Testers
CreateDate : 12/10/2014 3:39:11 PM
GroupId    : RHNZZGQJ7QHMAEXAMPLE3
GroupName  : Testers
Path       : /
```

- For API details, see ListGroups in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a collection of all the IAM groups defined in the current AWS account.

```
Get-IAMGroupList
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Administrators
CreateDate : 10/20/2014 10:06:24 AM
GroupId    : 6WCH4TRY3KIHIEXAMPLE1
GroupName  : Administrators
Path       : /

Arn        : arn:aws:iam::123456789012:group/Developers
CreateDate : 12/10/2014 3:38:55 PM
GroupId    : ZU2EOWMK6WBZOEXAMPLE2
GroupName  : Developers
Path       : /
```

```
Arn        : arn:aws:iam::123456789012:group/Testers
CreateDate : 12/10/2014 3:39:11 PM
GroupId    : RHNZZGQJ7QHMAEXAMPLE3
GroupName  : Testers
Path       : /
```

- For API details, see ListGroups in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_groups(count):
    """
    Lists the specified number of groups for the account.

    :param count: The number of groups to list.
    """
    try:
        for group in iam.groups.limit(count):
            logger.info("Group: %s", group.name)
    except ClientError:
        logger.exception("Couldn't list groups for the account.")
        raise
```

- For API details, see ListGroups in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# A class to manage IAM operations via the AWS SDK client
class IamGroupManager
  # Initializes the IamGroupManager class
  # @param iam_client [Aws::IAM::Client] An instance of the IAM client
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
  end

  # Lists up to a specified number of groups for the account.
  # @param count [Integer] The maximum number of groups to list.
  # @return [Aws::IAM::Client::Response]
  def list_groups(count)
    response = @iam_client.list_groups(max_items: count)
    response.groups.each do |group|
      @logger.info("\t#{group.group_name}")
    end
    response
  rescue Aws::Errors::ServiceError => e
    @logger.error("Couldn't list groups for the account. Here's why:")
    @logger.error("\t#{e.code}: #{e.message}")
    raise
  end
end
```

- For API details, see ListGroups in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn list_groups(
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListGroupsOutput, SdkError<ListGroupsError>> {
    let response = client
        .list_groups()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- For API details, see ListGroups in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listgroups( ).
        MESSAGE 'Retrieved group list.' TYPE 'I'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when listing groups.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListGroups in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func listGroups() async throws -> [String] {
        var groupList: [String] = []

        // Use "Paginated" to get all the groups.
        // This lets the SDK handle the 'isTruncated' property in
 "ListGroupsOutput".
        let input = ListGroupsInput()

        let pages = client.listGroupsPaginated(input: input)
        do {
            for try await page in pages {
                guard let groups = page.groups else {
                    print("Error: no groups returned.")
                    continue
                }

                for group in groups {
```

```
                    if let name = group.groupName {
                        groupList.append(name)
                    }
                }
            }
        } catch {
            print("ERROR: listGroups:", dump(error))
            throw error
        }
        return groupList
    }
```

- For API details, see ListGroups in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListGroupsForUser` with a CLI

The following code examples show how to use `ListGroupsForUser`.

CLI

AWS CLI

To list the groups that an IAM user belongs to

The following `list-groups-for-user` example displays the groups that the IAM user named Bob belongs to.

```
aws iam list-groups-for-user \
    --user-name Bob
```

Output:

```
{
    "Groups": [
        {
            "Path": "/",
            "CreateDate": "2013-05-06T01:18:08Z",
```

```
            "GroupId": "AKIAIOSFODNN7EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:group/Admin",
            "GroupName": "Admin"
        },
        {
            "Path": "/",
            "CreateDate": "2013-05-06T01:37:28Z",
            "GroupId": "AKIAI44QH8DHBEXAMPLE",
            "Arn": "arn:aws:iam::123456789012:group/s3-Users",
            "GroupName": "s3-Users"
        }
    ]
}
```

For more information, see Managing IAM user groups in the *AWS IAM User Guide*.

- For API details, see ListGroupsForUser in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns the list of IAM groups that the IAM user `David` belongs to.

```
Get-IAMGroupForUser -UserName David
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Administrators
CreateDate : 10/20/2014 10:06:24 AM
GroupId    : 6WCH4TRY3KIHIEXAMPLE1
GroupName  : Administrators
Path       : /

Arn        : arn:aws:iam::123456789012:group/Testers
CreateDate : 12/10/2014 3:39:11 PM
GroupId    : RHNZZGQJ7QHMAEXAMPLE2
GroupName  : Testers
Path       : /

Arn        : arn:aws:iam::123456789012:group/Developers
CreateDate : 12/10/2014 3:38:55 PM
```

```
GroupId    : ZU2EOWMK6WBZOEXAMPLE3
GroupName  : Developers
Path       : /
```

- For API details, see ListGroupsForUser in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns the list of IAM groups that the IAM user `David` belongs to.

```
Get-IAMGroupForUser -UserName David
```

Output:

```
Arn        : arn:aws:iam::123456789012:group/Administrators
CreateDate : 10/20/2014 10:06:24 AM
GroupId    : 6WCH4TRY3KIHIEXAMPLE1
GroupName  : Administrators
Path       : /

Arn        : arn:aws:iam::123456789012:group/Testers
CreateDate : 12/10/2014 3:39:11 PM
GroupId    : RHNZZGQJ7QHMAEXAMPLE2
GroupName  : Testers
Path       : /

Arn        : arn:aws:iam::123456789012:group/Developers
CreateDate : 12/10/2014 3:38:55 PM
GroupId    : ZU2EOWMK6WBZOEXAMPLE3
GroupName  : Developers
Path       : /
```

- For API details, see ListGroupsForUser in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListInstanceProfiles` with a CLI

The following code examples show how to use `ListInstanceProfiles`.

CLI

AWS CLI

To lists the instance profiles for the account

The following `list-instance-profiles` example lists the instance profiles that are associated with the current account.

```
aws iam list-instance-profiles
```

Output:

```
{
    "InstanceProfiles": [
        {
            "Path": "/",
            "InstanceProfileName": "example-dev-role",
            "InstanceProfileId": "AIPAIXEU4NUHUPEXAMPLE",
            "Arn": "arn:aws:iam::123456789012:instance-profile/example-dev-role",
            "CreateDate": "2023-09-21T18:17:41+00:00",
            "Roles": [
                {
                    "Path": "/",
                    "RoleName": "example-dev-role",
                    "RoleId": "AROAJ52OTH4H7LEXAMPLE",
                    "Arn": "arn:aws:iam::123456789012:role/example-dev-role",
                    "CreateDate": "2023-09-21T18:17:40+00:00",
                    "AssumeRolePolicyDocument": {
                        "Version":"2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Principal": {
                                    "Service": "ec2.amazonaws.com"
                                },
                                "Action": "sts:AssumeRole"
                            }
                        ]
                    }
                }
            ]
        },
```

```
        {
            "Path": "/",
            "InstanceProfileName": "example-s3-role",
            "InstanceProfileId": "AIPAJVJVNRIQFREXAMPLE",
            "Arn": "arn:aws:iam::123456789012:instance-profile/example-s3-role",
            "CreateDate": "2023-09-21T18:18:50+00:00",
            "Roles": [
                {
                    "Path": "/",
                    "RoleName": "example-s3-role",
                    "RoleId": "AROAINUBC5O7XLEXAMPLE",
                    "Arn": "arn:aws:iam::123456789012:role/example-s3-role",
                    "CreateDate": "2023-09-21T18:18:49+00:00",
                    "AssumeRolePolicyDocument": {
                        "Version":"2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Principal": {
                                    "Service": "ec2.amazonaws.com"
                                },
                                "Action": "sts:AssumeRole"
                            }
                        ]
                    }
                }
            ]
        }
    ]
}
```

For more information, see Using instance profiles in the *AWS IAM User Guide*.

- For API details, see ListInstanceProfiles in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a collection of the instance profiles defined in the current AWS account.

```
Get-IAMInstanceProfileList
```

Output:

```
Arn                 : arn:aws:iam::123456789012:instance-profile/ec2instancerole
CreateDate          : 2/17/2015 2:49:04 PM
InstanceProfileId   : HH36PTZQJUR32EXAMPLE1
InstanceProfileName : ec2instancerole
Path                : /
Roles               : {ec2instancerole}
```

- For API details, see ListInstanceProfiles in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a collection of the instance profiles defined in the current AWS account.

```
Get-IAMInstanceProfileList
```

Output:

```
Arn                 : arn:aws:iam::123456789012:instance-profile/ec2instancerole
CreateDate          : 2/17/2015 2:49:04 PM
InstanceProfileId   : HH36PTZQJUR32EXAMPLE1
InstanceProfileName : ec2instancerole
Path                : /
Roles               : {ec2instancerole}
```

- For API details, see ListInstanceProfiles in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListInstanceProfilesForRole` with a CLI

The following code examples show how to use `ListInstanceProfilesForRole`.

CLI

AWS CLI

To list the instance profiles for an IAM role

The following `list-instance-profiles-for-role` example lists the instance profiles that are associated with the role `Test-Role`.

```
aws iam list-instance-profiles-for-role \
    --role-name Test-Role
```

Output:

```
{
    "InstanceProfiles": [
        {
            "InstanceProfileId": "AIDGPMS9RO4H3FEXAMPLE",
            "Roles": [
                {
                    "AssumeRolePolicyDocument": "<URL-encoded-JSON>",
                    "RoleId": "AIDACKCEVSQ6C2EXAMPLE",
                    "CreateDate": "2013-06-07T20:42:15Z",
                    "RoleName": "Test-Role",
                    "Path": "/",
                    "Arn": "arn:aws:iam::123456789012:role/Test-Role"
                }
            ],
            "CreateDate": "2013-06-07T21:05:24Z",
            "InstanceProfileName": "ExampleInstanceProfile",
            "Path": "/",
            "Arn": "arn:aws:iam::123456789012:instance-profile/
ExampleInstanceProfile"
        }
    ]
}
```

For more information, see Using instance profiles in the *AWS IAM User Guide*.

- For API details, see ListInstanceProfilesForRole in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details of the instance profile associated with the role `ec2instancerole`.

```
Get-IAMInstanceProfileForRole -RoleName ec2instancerole
```

Output:

```
      Arn                 : arn:aws:iam::123456789012:instance-profile/
ec2instancerole
      CreateDate          : 2/17/2015 2:49:04 PM
      InstanceProfileId   : HH36PTZQJUR32EXAMPLE1
      InstanceProfileName : ec2instancerole
      Path                : /
      Roles               : {ec2instancerole}
```

- For API details, see ListInstanceProfilesForRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details of the instance profile associated with the role `ec2instancerole`.

```
Get-IAMInstanceProfileForRole -RoleName ec2instancerole
```

Output:

```
      Arn                 : arn:aws:iam::123456789012:instance-profile/
ec2instancerole
      CreateDate          : 2/17/2015 2:49:04 PM
      InstanceProfileId   : HH36PTZQJUR32EXAMPLE1
      InstanceProfileName : ec2instancerole
      Path                : /
      Roles               : {ec2instancerole}
```

- For API details, see ListInstanceProfilesForRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListMfaDevices` with a CLI

The following code examples show how to use `ListMfaDevices`.

CLI

AWS CLI

To list all MFA devices for a specified user

The following `list-mfa-devices` example returns details about the MFA device assigned to the IAM user Bob.

```
aws iam list-mfa-devices \
    --user-name Bob
```

Output:

```
{
    "MFADevices": [
        {
            "UserName": "Bob",
            "SerialNumber": "arn:aws:iam::123456789012:mfa/Bob",
            "EnableDate": "2019-10-28T20:37:09+00:00"
        },
        {
            "UserName": "Bob",
            "SerialNumber": "GAKT12345678",
            "EnableDate": "2023-02-18T21:44:42+00:00"
        },
        {
            "UserName": "Bob",
            "SerialNumber": "arn:aws:iam::123456789012:u2f/user/Bob/
fidosecuritykey1-7XNL7NFNLZ123456789EXAMPLE",
            "EnableDate": "2023-09-19T02:25:35+00:00"
        },
        {
            "UserName": "Bob",
            "SerialNumber": "arn:aws:iam::123456789012:u2f/user/Bob/
fidosecuritykey2-VDRQTDBBN5123456789EXAMPLE",
            "EnableDate": "2023-09-19T01:49:18+00:00"
        }
    ]
```

```
}
```

For more information, see Using multi-factor authentication (MFA) in AWS in the *AWS IAM User Guide*.

- For API details, see ListMfaDevices in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns details about the MFA device assigned to the IAM user `David`. In this example you can tell that it is a virtual device because the `SerialNumber` is an ARN instead of a physical device's actual serial number.

```
Get-IAMMFADevice -UserName David
```

Output:

```
EnableDate                SerialNumber                            UserName
----------                ------------                            --------
4/8/2015 9:41:10 AM       arn:aws:iam::123456789012:mfa/David     David
```

- For API details, see ListMfaDevices in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns details about the MFA device assigned to the IAM user `David`. In this example you can tell that it is a virtual device because the `SerialNumber` is an ARN instead of a physical device's actual serial number.

```
Get-IAMMFADevice -UserName David
```

Output:

```
EnableDate                SerialNumber                            UserName
----------                ------------                            --------
4/8/2015 9:41:10 AM       arn:aws:iam::123456789012:mfa/David     David
```

- For API details, see ListMfaDevices in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListOpenIdConnectProviders` with a CLI

The following code examples show how to use `ListOpenIdConnectProviders`.

CLI

AWS CLI

To list information about the OpenID Connect providers in the AWS account

The following `list-open-id-connect-providers` example returns a list of ARNs of all the OpenID Connect providers that are defined in the current AWS account.

```
aws iam list-open-id-connect-providers
```

Output:

```
{
    "OpenIDConnectProviderList": [
        {
            "Arn": "arn:aws:iam::123456789012:oidc-provider/
example.oidcprovider.com"
        }
    ]
}
```

For more information, see Creating OpenID Connect (OIDC) identity providers in the *AWS IAM User Guide*.

- For API details, see ListOpenIdConnectProviders in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a list of ARNS of all the OpenID Connect providers that are defined in the current AWS account.

```
Get-IAMOpenIDConnectProviderList
```

Output:

```
Arn
---
arn:aws:iam::123456789012:oidc-provider/server.example.com
arn:aws:iam::123456789012:oidc-provider/another.provider.com
```

- For API details, see ListOpenIdConnectProviders in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a list of ARNS of all the OpenID Connect providers that are defined in the current AWS account.

```
Get-IAMOpenIDConnectProviderList
```

Output:

```
Arn
---
arn:aws:iam::123456789012:oidc-provider/server.example.com
arn:aws:iam::123456789012:oidc-provider/another.provider.com
```

- For API details, see ListOpenIdConnectProviders in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListPolicies` with an AWS SDK or CLI

The following code examples show how to use `ListPolicies`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage policies

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }
```

- For API details, see ListPolicies in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::listPolicies(const Aws::Client::ClientConfiguration
 &clientConfig) {
    const Aws::String DATE_FORMAT("%Y-%m-%d");
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListPoliciesRequest request;

    bool done = false;
    bool header = false;
    while (!done) {
        auto outcome = iam.ListPolicies(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list iam policies: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        if (!header) {
            std::cout << std::left << std::setw(55) << "Name" <<
                      std::setw(30) << "ID" << std::setw(80) << "Arn" <<
                      std::setw(64) << "Description" << std::setw(12) <<
                      "CreateDate" << std::endl;
            header = true;
        }

        const auto &policies = outcome.GetResult().GetPolicies();
        for (const auto &policy: policies) {
            std::cout << std::left << std::setw(55) <<
                      policy.GetPolicyName() << std::setw(30) <<
                      policy.GetPolicyId() << std::setw(80) << policy.GetArn() <<
                      std::setw(64) << policy.GetDescription() << std::setw(12)
 <<
                      policy.GetCreateDate().ToGmtString(DATE_FORMAT.c_str()) <<
                      std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }
```

```
    return true;
}
```

- For API details, see ListPolicies in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list managed policies that are available to your AWS account

The following `list-policies` example returns a collection of the first three managed policies available in the current AWS account.

```
aws iam list-policies \
    --max-items 3
```

Output:

```
{
    "Policies": [
        {
            "PolicyName": "AWSCloudTrailAccessPolicy",
            "PolicyId": "ANPAXQE2B5PJ7YEXAMPLE",
            "Arn": "arn:aws:iam::123456789012:policy/AWSCloudTrailAccessPolicy",
            "Path": "/",
            "DefaultVersionId": "v1",
            "AttachmentCount": 0,
            "PermissionsBoundaryUsageCount": 0,
            "IsAttachable": true,
            "CreateDate": "2019-09-04T17:43:42+00:00",
            "UpdateDate": "2019-09-04T17:43:42+00:00"
        },
        {
            "PolicyName": "AdministratorAccess",
            "PolicyId": "ANPAIWMBCKSKIEE64ZLYK",
            "Arn": "arn:aws:iam::aws:policy/AdministratorAccess",
            "Path": "/",
            "DefaultVersionId": "v1",
            "AttachmentCount": 6,
            "PermissionsBoundaryUsageCount": 0,
```

```
            "IsAttachable": true,
            "CreateDate": "2015-02-06T18:39:46+00:00",
            "UpdateDate": "2015-02-06T18:39:46+00:00"
        },
        {
            "PolicyName": "PowerUserAccess",
            "PolicyId": "ANPAJYRXTHIB4FOVS3ZXS",
            "Arn": "arn:aws:iam::aws:policy/PowerUserAccess",
            "Path": "/",
            "DefaultVersionId": "v5",
            "AttachmentCount": 1,
            "PermissionsBoundaryUsageCount": 0,
            "IsAttachable": true,
            "CreateDate": "2015-02-06T18:39:47+00:00",
            "UpdateDate": "2023-07-06T22:04:00+00:00"
        }
    ],
    "NextToken": "EXAMPLErZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiA4fQ=="
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see ListPolicies in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
```

```
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(ctx context.Context, maxPolicies int32)
 ([]types.Policy, error) {
 var policies []types.Policy
 result, err := wrapper.IamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPolicies),
 })
 if err != nil {
  log.Printf("Couldn't list policies. Here's why: %v\n", err)
 } else {
  policies = result.Policies
 }
 return policies, err
}
```

- For API details, see ListPolicies in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the policies.

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
 *
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services
 account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListPoliciesCommand({
          Marker: response.Marker,
          MaxItems: 10,
          OnlyAttached: false,
          Scope: "Local",
        }),
      );
    } else {
      break;
    }
  }
}
```

- For API details, see ListPolicies in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listPolicies($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listPoliciesArguments = [];
        if ($pathPrefix) {
            $listPoliciesArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listPoliciesArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listPoliciesArguments["MaxItems"] = $maxItems;
        }

        return $this->iamClient->listPolicies($listPoliciesArguments);
    }
```

- For API details, see ListPolicies in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a collection of the first three managed policies available in the current AWS account. Because `-scope` is not specified, it defaults to `all` and includes both AWS managed and customer managed policies.

```
Get-IAMPolicyList -MaxItem 3
```

Output:

```
Arn              : arn:aws:iam::aws:policy/AWSDirectConnectReadOnlyAccess
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:08 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : Z27SI6FQMGNQ2EXAMPLE1
PolicyName       : AWSDirectConnectReadOnlyAccess
UpdateDate       : 2/6/2015 10:40:08 AM

Arn              : arn:aws:iam::aws:policy/AmazonGlacierReadOnlyAccess
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:27 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : NJKMU274MET4EEXAMPLE2
PolicyName       : AmazonGlacierReadOnlyAccess
UpdateDate       : 2/6/2015 10:40:27 AM

Arn              : arn:aws:iam::aws:policy/AWSMarketplaceFullAccess
AttachmentCount  : 0
CreateDate       : 2/11/2015 9:21:45 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : 5ULJSO2FYVPYGEXAMPLE3
PolicyName       : AWSMarketplaceFullAccess
```

```
UpdateDate       : 2/11/2015 9:21:45 AM
```

Example 2: This example returns a collection of the first two customer managed policies available in current AWS account. It uses `-Scope local` to limit the output to only customer managed policies.

```
Get-IAMPolicyList -Scope local -MaxItem 2
```

Output:

```
Arn              : arn:aws:iam::123456789012:policy/MyLocalPolicy
AttachmentCount  : 0
CreateDate       : 2/12/2015 9:39:09 AM
DefaultVersionId : v2
Description      :
IsAttachable     : True
Path             : /
PolicyId         : SQVCBLC4VAOUCEXAMPLE4
PolicyName       : MyLocalPolicy
UpdateDate       : 2/12/2015 9:39:53 AM

Arn              : arn:aws:iam::123456789012:policy/policyforec2instancerole
AttachmentCount  : 1
CreateDate       : 2/17/2015 2:51:38 PM
DefaultVersionId : v11
Description      :
IsAttachable     : True
Path             : /
PolicyId         : X5JPBLJH2Z2SOEXAMPLE5
PolicyName       : policyforec2instancerole
UpdateDate       : 2/18/2015 8:52:31 AM
```

- For API details, see ListPolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a collection of the first three managed policies available in the current AWS account. Because `-scope` is not specified, it defaults to `all` and includes both AWS managed and customer managed policies.

```
Get-IAMPolicyList -MaxItem 3
```

Output:

```
Arn              : arn:aws:iam::aws:policy/AWSDirectConnectReadOnlyAccess
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:08 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : Z27SI6FQMGNQ2EXAMPLE1
PolicyName       : AWSDirectConnectReadOnlyAccess
UpdateDate       : 2/6/2015 10:40:08 AM

Arn              : arn:aws:iam::aws:policy/AmazonGlacierReadOnlyAccess
AttachmentCount  : 0
CreateDate       : 2/6/2015 10:40:27 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : NJKMU274MET4EEXAMPLE2
PolicyName       : AmazonGlacierReadOnlyAccess
UpdateDate       : 2/6/2015 10:40:27 AM

Arn              : arn:aws:iam::aws:policy/AWSMarketplaceFullAccess
AttachmentCount  : 0
CreateDate       : 2/11/2015 9:21:45 AM
DefaultVersionId : v1
Description      :
IsAttachable     : True
Path             : /
PolicyId         : 5ULJSO2FYVPYGEXAMPLE3
PolicyName       : AWSMarketplaceFullAccess
UpdateDate       : 2/11/2015 9:21:45 AM
```

Example 2: This example returns a collection of the first two customer managed policies available in current AWS account. It uses `-Scope local` to limit the output to only customer managed policies.

```
Get-IAMPolicyList -Scope local -MaxItem 2
```

Output:

```
Arn              : arn:aws:iam::123456789012:policy/MyLocalPolicy
AttachmentCount  : 0
CreateDate       : 2/12/2015 9:39:09 AM
DefaultVersionId : v2
Description      :
IsAttachable     : True
Path             : /
PolicyId         : SQVCBLC4VAOUCEXAMPLE4
PolicyName       : MyLocalPolicy
UpdateDate       : 2/12/2015 9:39:53 AM

Arn              : arn:aws:iam::123456789012:policy/policyforec2instancerole
AttachmentCount  : 1
CreateDate       : 2/17/2015 2:51:38 PM
DefaultVersionId : v11
Description      :
IsAttachable     : True
Path             : /
PolicyId         : X5JPBLJH2Z2SOEXAMPLE5
PolicyName       : policyforec2instancerole
UpdateDate       : 2/18/2015 8:52:31 AM
```

- For API details, see ListPolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_policies(scope):
    """
    Lists the policies in the current account.

    :param scope: Limits the kinds of policies that are returned. For example,
                  'Local' specifies that only locally managed policies are
 returned.
```

```
    :return: The list of policies.
    """
    try:
        policies = list(iam.policies.filter(Scope=scope))
        logger.info("Got %s policies in scope '%s'.", len(policies), scope)
    except ClientError:
        logger.exception("Couldn't get policies for scope '%s'.", scope)
        raise
    else:
        return policies
```

- For API details, see ListPolicies in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example module lists, creates, attaches, and detaches role policies.

```
# Manages policies in AWS Identity and Access Management (IAM)
class RolePolicyManager
  # Initialize with an AWS IAM client
  #
  # @param iam_client [Aws::IAM::Client] An initialized IAM client
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'PolicyManager'
  end

  # Creates a policy
  #
  # @param policy_name [String] The name of the policy
```

```
  # @param policy_document [Hash] The policy document
  # @return [String] The policy ARN if successful, otherwise nil
  def create_policy(policy_name, policy_document)
    response = @iam_client.create_policy(
      policy_name: policy_name,
      policy_document: policy_document.to_json
    )
    response.policy.arn
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error creating policy: #{e.message}")
    nil
  end

  # Fetches an IAM policy by its ARN
  # @param policy_arn [String] the ARN of the IAM policy to retrieve
  # @return [Aws::IAM::Types::GetPolicyResponse] the policy object if found
  def get_policy(policy_arn)
    response = @iam_client.get_policy(policy_arn: policy_arn)
    policy = response.policy
    @logger.info("Got policy '#{policy.policy_name}'. Its ID is:
#{policy.policy_id}.")
    policy
  rescue Aws::IAM::Errors::NoSuchEntity
    @logger.error("Couldn't get policy '#{policy_arn}'. The policy does not
exist.")
    raise
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't get policy '#{policy_arn}'. Here's why: #{e.code}:
#{e.message}")
    raise
  end

  # Attaches a policy to a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def attach_policy_to_role(role_name, policy_arn)
    @iam_client.attach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
```

```
    @logger.error("Error attaching policy to role: #{e.message}")
    false
  end

  # Lists policy ARNs attached to a role
  #
  # @param role_name [String] The name of the role
  # @return [Array<String>] List of policy ARNs
  def list_attached_policy_arns(role_name)
    response = @iam_client.list_attached_role_policies(role_name: role_name)
    response.attached_policies.map(&:policy_arn)
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing policies attached to role: #{e.message}")
    []
  end

  # Detaches a policy from a role
  #
  # @param role_name [String] The name of the role
  # @param policy_arn [String] The policy ARN
  # @return [Boolean] true if successful, false otherwise
  def detach_policy_from_role(role_name, policy_arn)
    @iam_client.detach_role_policy(
      role_name: role_name,
      policy_arn: policy_arn
    )
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error detaching policy from role: #{e.message}")
    false
  end
end
```

- For API details, see ListPolicies in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn list_policies(
    client: iamClient,
    path_prefix: String,
) -> Result<Vec<String>, SdkError<ListPoliciesError>> {
    let list_policies = client
        .list_policies()
        .path_prefix(path_prefix)
        .scope(PolicyScopeType::Local)
        .into_paginator()
        .items()
        .send()
        .try_collect()
        .await?;

    let policy_names = list_policies
        .into_iter()
        .map(|p| {
            let name = p
                .policy_name
                .unwrap_or_else(|| "Missing Policy Name".to_string());
            println!("{}", name);
            name
        })
        .collect();

    Ok(policy_names)
}
```

- For API details, see ListPolicies in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listpolicies( iv_scope = iv_scope ).
        MESSAGE 'Retrieved policy list.' TYPE 'I'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when listing policies.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListPolicies in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func listPolicies() async throws -> [MyPolicyRecord] {
        var policyList: [MyPolicyRecord] = []

        // Use "Paginated" to get all the policies.
        // This lets the SDK handle the 'isTruncated' in "ListPoliciesOutput".
        let input = ListPoliciesInput()
```

```
        let output = client.listPoliciesPaginated(input: input)

        do {
            for try await page in output {
                guard let policies = page.policies else {
                    print("Error: no policies returned.")
                    continue
                }

                for policy in policies {
                    guard let name = policy.policyName,
                          let id = policy.policyId,
                          let arn = policy.arn
                    else {
                        throw ServiceHandlerError.noSuchPolicy
                    }
                    policyList.append(MyPolicyRecord(name: name, id: id, arn:
arn))
                }
            }
        } catch {
            print("ERROR: listPolicies:", dump(error))
            throw error
        }

        return policyList
    }
```

- For API details, see ListPolicies in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListPolicyVersions` with an AWS SDK or CLI

The following code examples show how to use `ListPolicyVersions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Manage policies

- Roll back a policy version

CLI

AWS CLI

To list information about the versions of the specified managed policy

The following `list-policy-versions` example returns the list of available versions of the policy whose ARN is `arn:aws:iam::123456789012:policy/MySamplePolicy`.

```
aws iam list-policy-versions \
    --policy-arn arn:aws:iam::123456789012:policy/MySamplePolicy
```

Output:

```
{
    "IsTruncated": false,
    "Versions": [
        {
        "VersionId": "v2",
        "IsDefaultVersion": true,
        "CreateDate": "2015-06-02T23:19:44Z"
        },
        {
        "VersionId": "v1",
        "IsDefaultVersion": false,
        "CreateDate": "2015-06-02T22:30:47Z"
        }
    ]
}
```

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see ListPolicyVersions in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns the list of available versions of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyManagedPolicy`. To get the policy

document for a specific version, use the `Get-IAMPolicyVersion` command and specify the `VersionId` of the one you want.

```
Get-IAMPolicyVersionList -PolicyArn arn:aws:iam::123456789012:policy/
MyManagedPolicy
```

Output:

```
CreateDate                  Document                IsDefaultVersion
       VersionId
----------                  --------                ---------------
       ---------
2/12/2015 9:39:53 AM                                True
       v2
2/12/2015 9:39:09 AM                                False
       v1
```

- For API details, see ListPolicyVersions in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns the list of available versions of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyManagedPolicy`. To get the policy document for a specific version, use the `Get-IAMPolicyVersion` command and specify the `VersionId` of the one you want.

```
Get-IAMPolicyVersionList -PolicyArn arn:aws:iam::123456789012:policy/
MyManagedPolicy
```

Output:

```
CreateDate                  Document                IsDefaultVersion
       VersionId
----------                  --------                ---------------
       ---------
2/12/2015 9:39:53 AM                                True
       v2
2/12/2015 9:39:09 AM                                False
       v1
```

- For API details, see ListPolicyVersions in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_iam->listpolicyversions(
      iv_policyarn = iv_policy_arn ).
    MESSAGE 'Retrieved policy versions list.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'Policy does not exist.' TYPE 'E'.
  CATCH /aws1/cx_iamservicefailureex.
    MESSAGE 'Service failure when listing policy versions.' TYPE 'E'.
ENDTRY.
```

- For API details, see ListPolicyVersions in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListRolePolicies` with an AWS SDK or CLI

The following code examples show how to use `ListRolePolicies`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Getting started with internet of things device protection

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</
param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator =
 _IAMService.Paginators.ListRolePolicies(new ListRolePoliciesRequest { RoleName =
 roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }
```

- For API details, see ListRolePolicies in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list the policies attached to an IAM role

The following `list-role-policies` example lists the names of the permissions policies for the specified IAM role.

```
aws iam list-role-policies \
    --role-name Test-Role
```

Output:

```
{
    "PolicyNames": [
        "ExamplePolicy"
    ]
}
```

To see the trust policy attached to a role, use the `get-role` command. To see the details of a permissions policy, use the `get-role-policy` command.

For more information, see Creating IAM roles in the *AWS IAM User Guide*.

- For API details, see ListRolePolicies in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)
```

```
// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(ctx context.Context, roleName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListRolePolicies(ctx,
 &iam.ListRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName,
 err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}
```

- For API details, see ListRolePolicies in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the policies.

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
 *
 * @param {string} roleName
 */
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- For API details, see ListRolePolicies in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listRolePolicies($roleName, $marker = "", $maxItems = 0)
    {
        $listRolePoliciesArguments = ['RoleName' => $roleName];
        if ($marker) {
            $listRolePoliciesArguments['Marker'] = $marker;
        }
        if ($maxItems) {
            $listRolePoliciesArguments['MaxItems'] = $maxItems;
        }
        return $this->customWaiter(function () use ($listRolePoliciesArguments) {
            return $this->iamClient-
>listRolePolicies($listRolePoliciesArguments);
        });
    }
```

- For API details, see ListRolePolicies in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns the list of names of inline policies that are embedded in the IAM role `lamda_exec_role`. To see the details of an inline policy, use the command `Get-IAMRolePolicy`.

```
Get-IAMRolePolicyList -RoleName lambda_exec_role
```

Output:

```
oneClick_lambda_exec_role_policy
```

- For API details, see ListRolePolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns the list of names of inline policies that are embedded in the IAM role `lamda_exec_role`. To see the details of an inline policy, use the command `Get-IAMRolePolicy`.

```
Get-IAMRolePolicyList -RoleName lambda_exec_role
```

Output:

```
oneClick_lambda_exec_role_policy
```

- For API details, see ListRolePolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_policies(role_name):
    """
    Lists inline policies for a role.

    :param role_name: The name of the role to query.
    """
    try:
        role = iam.Role(role_name)
        for policy in role.policies.all():
```

```
            logger.info("Got inline policy %s.", policy.name)
    except ClientError:
        logger.exception("Couldn't list inline policies for %s.", role_name)
        raise
```

- For API details, see ListRolePolicies in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Lists policy ARNs attached to a role
#
# @param role_name [String] The name of the role
# @return [Array<String>] List of policy ARNs
def list_attached_policy_arns(role_name)
  response = @iam_client.list_attached_role_policies(role_name: role_name)
  response.attached_policies.map(&:policy_arn)
rescue Aws::IAM::Errors::ServiceError => e
  @logger.error("Error listing policies attached to role: #{e.message}")
  []
end
```

- For API details, see ListRolePolicies in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn list_role_policies(
    client: &iamClient,
    role_name: &str,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListRolePoliciesOutput, SdkError<ListRolePoliciesError>> {
    let response = client
        .list_role_policies()
        .role_name(role_name)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;

    Ok(response)
}
```

- For API details, see ListRolePolicies in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listrolepolicies(
          iv_rolename = iv_role_name ).
        MESSAGE 'Retrieved inline policy list for role.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Role does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListRolePolicies in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func listRolePolicies(role: String) async throws -> [String] {
        var policyList: [String] = []

        // Use "Paginated" to get all the role policies.
        // This lets the SDK handle the 'isTruncated' in
 "ListRolePoliciesOutput".
        let input = ListRolePoliciesInput(
            roleName: role
        )
        let pages = client.listRolePoliciesPaginated(input: input)

        do {
            for try await page in pages {
                guard let policies = page.policyNames else {
                    print("Error: no role policies returned.")
                    continue
```

```
                }

                for policy in policies {
                    policyList.append(policy)
                }
            }
        } catch {
            print("ERROR: listRolePolicies:", dump(error))
            throw error
        }
        return policyList
    }
```

- For API details, see ListRolePolicies in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListRoleTags` with a CLI

The following code examples show how to use `ListRoleTags`.

CLI

AWS CLI

To list the tags attached to a role

The following `list-role-tags` example retrieves the list of tags associated with the specified role.

```
aws iam list-role-tags \
    --role-name production-role
```

Output:

```
{
    "Tags": [
        {
```

```
            "Key": "Department",
            "Value": "Accounting"
        },
        {
            "Key": "DeptID",
            "Value": "12345"
        }
    ],
    "IsTruncated": false
}
```

For more information, see Tagging IAM resources in the *AWS IAM User Guide*.

- For API details, see ListRoleTags in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example fetches the tag associated with the role..

```
Get-IAMRoleTagList -RoleName MyRoleName
```

- For API details, see ListRoleTags in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example fetches the tag associated with the role..

```
Get-IAMRoleTagList -RoleName MyRoleName
```

- For API details, see ListRoleTags in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListRoles` with an AWS SDK or CLI

The following code examples show how to use `ListRoles`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }
```

- For API details, see ListRoles in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list IAM roles for the current account

The following `list-roles` example lists IAM roles for the current account.

```
aws iam list-roles
```

Output:

```
{
    "Roles": [
        {
            "Path": "/",
            "RoleName": "ExampleRole",
            "RoleId": "AROAJ52OTH4H7LEXAMPLE",
            "Arn": "arn:aws:iam::123456789012:role/ExampleRole",
            "CreateDate": "2017-09-12T19:23:36+00:00",
            "AssumeRolePolicyDocument": {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Sid": "",
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "ec2.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "MaxSessionDuration": 3600
        },
        {
            "Path": "/example_path/",
            "RoleName": "ExampleRoleWithPath",
            "RoleId": "AROAI4QRP7UFT7EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:role/example_path/
ExampleRoleWithPath",
            "CreateDate": "2023-09-21T20:29:38+00:00",
            "AssumeRolePolicyDocument": {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Sid": "",
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "ec2.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
```

```
            },
            "MaxSessionDuration": 3600
        }
    ]
}
```

For more information, see Creating IAM roles in the *AWS IAM User Guide*.

- For API details, see ListRoles in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRoles gets up to maxRoles roles.
```

```
func (wrapper RoleWrapper) ListRoles(ctx context.Context, maxRoles int32)
 ([]types.Role, error) {
 var roles []types.Role
 result, err := wrapper.IamClient.ListRoles(ctx,
  &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
 )
 if err != nil {
  log.Printf("Couldn't list roles. Here's why: %v\n", err)
 } else {
  roles = result.Roles
 }
 return roles, err
}
```

- For API details, see ListRoles in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the roles.

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
 *
 */
export async function* listRoles() {
```

```
  const command = new ListRolesCommand({
    MaxItems: 10,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.Roles?.length) {
    for (const role of response.Roles) {
      yield role;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolesCommand({
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- For API details, see ListRoles in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();
```

```
    /**
     * @param string $pathPrefix
     * @param string $marker
     * @param int $maxItems
     * @return Result
     * $roles = $service->listRoles();
     */
    public function listRoles($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listRolesArguments = [];
        if ($pathPrefix) {
            $listRolesArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listRolesArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listRolesArguments["MaxItems"] = $maxItems;
        }
        return $this->iamClient->listRoles($listRolesArguments);
    }
```

- For API details, see ListRoles in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves a list of all of the IAM roles in the AWS account.

```
Get-IAMRoleList
```

- For API details, see ListRoles in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves a list of all of the IAM roles in the AWS account.

```
Get-IAMRoleList
```

- For API details, see ListRoles in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_roles(count):
    """
    Lists the specified number of roles for the account.

    :param count: The number of roles to list.
    """
    try:
        roles = list(iam.roles.limit(count=count))
        for role in roles:
            logger.info("Role: %s", role.name)
    except ClientError:
        logger.exception("Couldn't list roles for the account.")
        raise
    else:
        return roles
```

- For API details, see ListRoles in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
  # Lists IAM roles up to a specified count.
  # @param count [Integer] the maximum number of roles to list.
  # @return [Array<String>] the names of the roles.
  def list_roles(count)
    role_names = []
    roles_counted = 0

    @iam_client.list_roles.each_page do |page|
      page.roles.each do |role|
        break if roles_counted >= count

        @logger.info("\t#{roles_counted + 1}: #{role.role_name}")
        role_names << role.role_name
        roles_counted += 1
      end
      break if roles_counted >= count
    end

    role_names
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't list roles for the account. Here's why:")
    @logger.error("\t#{e.code}: #{e.message}")
    raise
  end
```

- For API details, see ListRoles in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn list_roles(
    client: &iamClient,
    path_prefix: Option<String>,
```

```
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListRolesOutput, SdkError<ListRolesError>> {
    let response = client
        .list_roles()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;
    Ok(response)
}
```

- For API details, see ListRoles in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listroles( ).
        MESSAGE 'Retrieved role list.' TYPE 'I'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when listing roles.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListRoles in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func listRoles() async throws -> [String] {
        var roleList: [String] = []

        // Use "Paginated" to get all the roles.
        // This lets the SDK handle the 'isTruncated' in "ListRolesOutput".
        let input = ListRolesInput()
        let pages = client.listRolesPaginated(input: input)

        do {
            for try await page in pages {
                guard let roles = page.roles else {
                    print("Error: no roles returned.")
                    continue
                }

                for role in roles {
                    if let name = role.roleName {
                        roleList.append(name)
                    }
                }
            }
        } catch {
            print("ERROR: listRoles:", dump(error))
            throw error
        }
        return roleList
    }
```

- For API details, see ListRoles in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListSAMLProviders` with an AWS SDK or CLI

The following code examples show how to use `ListSAMLProviders`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }
```

- For API details, see ListSAMLProviders in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list the SAML providers in the AWS account

The following `list-saml-providers` example retrieves the list of SAML 2.0 providers created in the current AWS account.

```
aws iam list-saml-providers
```

Output:

```
{
    "SAMLProviderList": [
        {
            "Arn": "arn:aws:iam::123456789012:saml-provider/SAML-ADFS",
            "ValidUntil": "2015-06-05T22:45:14Z",
            "CreateDate": "2015-06-05T22:45:14Z"
        }
    ]
}
```

For more information, see Creating IAM SAML identity providers in the *AWS IAM User Guide*.

- For API details, see ListSAMLProviders in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)
```

```
// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders(ctx context.Context)
 ([]types.SAMLProviderListEntry, error) {
 var providers []types.SAMLProviderListEntry
 result, err := wrapper.IamClient.ListSAMLProviders(ctx,
 &iam.ListSAMLProvidersInput{})
 if err != nil {
  log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
 } else {
  providers = result.SAMLProviderList
 }
 return providers, err
}
```

- For API details, see ListSAMLProviders in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the SAML providers.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For API details, see ListSAMLProviders in *AWS SDK for JavaScript API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listSAMLProviders()
    {
        return $this->iamClient->listSAMLProviders();
    }
```

- For API details, see ListSAMLProviders in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves the list of SAML 2.0 providers created in the current AWS account. It returns the ARN, creation date, and expiration date for each SAML provider.

```
Get-IAMSAMLProviderList
```

Output:

```
Arn                                                 CreateDate
   ValidUntil
---                                                 ----------
   ----------
arn:aws:iam::123456789012:saml-provider/SAMLADFS    12/23/2014 12:16:55 PM
   12/23/2114 12:16:54 PM
```

- For API details, see ListSAMLProviders in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves the list of SAML 2.0 providers created in the current AWS account. It returns the ARN, creation date, and expiration date for each SAML provider.

```
Get-IAMSAMLProviderList
```

Output:

```
Arn                                                 CreateDate
   ValidUntil
---                                                 ----------
   ----------
arn:aws:iam::123456789012:saml-provider/SAMLADFS    12/23/2014 12:16:55 PM
   12/23/2114 12:16:54 PM
```

- For API details, see ListSAMLProviders in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_saml_providers(count):
    """
    Lists the SAML providers for the account.

    :param count: The maximum number of providers to list.
    """
    try:
        found = 0
        for provider in iam.saml_providers.limit(count):
            logger.info("Got SAML provider %s.", provider.arn)
            found += 1
        if found == 0:
            logger.info("Your account has no SAML providers.")
    except ClientError:
        logger.exception("Couldn't list SAML providers.")
        raise
```

- For API details, see ListSAMLProviders in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class SamlProviderLister
  # Initializes the SamlProviderLister with IAM client and a logger.
  # @param iam_client [Aws::IAM::Client] The IAM client object.
  # @param logger [Logger] The logger object for logging output.
  def initialize(iam_client, logger = Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
  end

  # Lists up to a specified number of SAML providers for the account.
  # @param count [Integer] The maximum number of providers to list.
  # @return [Aws::IAM::Client::Response]
  def list_saml_providers(count)
    response = @iam_client.list_saml_providers
    response.saml_provider_list.take(count).each do |provider|
      @logger.info("\t#{provider.arn}")
    end
    response
  rescue Aws::Errors::ServiceError => e
    @logger.error("Couldn't list SAML providers. Here's why:")
    @logger.error("\t#{e.code}: #{e.message}")
    raise
  end
end
```

- For API details, see ListSAMLProviders in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
```

```
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- For API details, see ListSAMLProviders in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listsamlproviders( ).
        MESSAGE 'Retrieved SAML provider list.' TYPE 'I'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when listing SAML providers.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListSAMLProviders in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListServerCertificates` with an AWS SDK or CLI

The following code examples show how to use `ListServerCertificates`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::listServerCertificates(
        const Aws::Client::ClientConfiguration &clientConfig) {
    const Aws::String DATE_FORMAT = "%Y-%m-%d";

    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListServerCertificatesRequest request;

    bool done = false;
    bool header = false;
    while (!done) {
        auto outcome = iam.ListServerCertificates(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list server certificates: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        if (!header) {
            std::cout << std::left << std::setw(55) << "Name" <<
                      std::setw(30) << "ID" << std::setw(80) << "Arn" <<
                      std::setw(14) << "UploadDate" << std::setw(14) <<
                      "ExpirationDate" << std::endl;
            header = true;
        }

        const auto &certificates =
                outcome.GetResult().GetServerCertificateMetadataList();

        for (const auto &certificate: certificates) {
            std::cout << std::left << std::setw(55) <<
                      certificate.GetServerCertificateName() << std::setw(30) <<
                      certificate.GetServerCertificateId() << std::setw(80) <<
```

```
                    certificate.GetArn() << std::setw(14) <<

 certificate.GetUploadDate().ToGmtString(DATE_FORMAT.c_str()) <<
                    std::setw(14) <<

 certificate.GetExpiration().ToGmtString(DATE_FORMAT.c_str()) <<
                    std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }

    return true;
}
```

- For API details, see ListServerCertificates in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list the server certificates in your AWS account

The following `list-server-certificates` example lists all of the server certificates stored and available for use in your AWS account.

```
aws iam list-server-certificates
```

Output:

```
{
    "ServerCertificateMetadataList": [
        {
            "Path": "/",
            "ServerCertificateName": "myUpdatedServerCertificate",
            "ServerCertificateId": "ASCAEXAMPLE123EXAMPLE",
```

```
            "Arn": "arn:aws:iam::123456789012:server-certificate/
myUpdatedServerCertificate",
            "UploadDate": "2019-04-22T21:13:44+00:00",
            "Expiration": "2019-10-15T22:23:16+00:00"
        },
        {
            "Path": "/cloudfront/",
            "ServerCertificateName": "MyTestCert",
            "ServerCertificateId": "ASCAEXAMPLE456EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:server-certificate/Org1/Org2/
MyTestCert",
            "UploadDate": "2015-04-21T18:14:16+00:00",
            "Expiration": "2018-01-14T17:52:36+00:00"
        }
    ]
}
```

For more information, see Managing server certificates in IAM in the *AWS IAM User Guide*.

- For API details, see ListServerCertificates in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the certificates.

```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to
 simplify this.
```

```
 *
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);

  while (response.ServerCertificateMetadataList?.length) {
    for await (const cert of response.ServerCertificateMetadataList) {
      yield cert;
    }

    if (response.IsTruncated) {
      response = await client.send(new ListServerCertificatesCommand({}));
    } else {
      break;
    }
  }
}
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListServerCertificates in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListServerCertificates in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves the list of server certificates that have been uploaded to the current AWS account.

```
Get-IAMServerCertificateList
```

Output:

```
Arn                   : arn:aws:iam::123456789012:server-certificate/Org1/Org2/
MyServerCertificate
Expiration            : 1/14/2018 9:52:36 AM
Path                  : /Org1/Org2/
ServerCertificateId   : ASCAJIFEXAMPLE17HQZYW
ServerCertificateName : MyServerCertificate
UploadDate            : 4/21/2015 11:14:16 AM
```

- For API details, see ListServerCertificates in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves the list of server certificates that have been uploaded to the current AWS account.

```
Get-IAMServerCertificateList
```

Output:

```
Arn                  : arn:aws:iam::123456789012:server-certificate/Org1/Org2/
MyServerCertificate
Expiration           : 1/14/2018 9:52:36 AM
Path                 : /Org1/Org2/
ServerCertificateId  : ASCAJIFEXAMPLE17HQZYW
ServerCertificateName : MyServerCertificate
UploadDate           : 4/21/2015 11:14:16 AM
```

- For API details, see ListServerCertificates in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List, update, and delete server certificates.

```
class ServerCertificateManager
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'ServerCertificateManager'
  end

  # Creates a new server certificate.
  # @param name [String] the name of the server certificate
  # @param certificate_body [String] the contents of the certificate
  # @param private_key [String] the private key contents
  # @return [Boolean] returns true if the certificate was successfully created
  def create_server_certificate(name, certificate_body, private_key)
    @iam_client.upload_server_certificate({
                                            server_certificate_name: name,
                                            certificate_body: certificate_body,
                                            private_key: private_key
                                          })
```

```
    true
  rescue Aws::IAM::Errors::ServiceError => e
    puts "Failed to create server certificate: #{e.message}"
    false
  end

  # Lists available server certificate names.
  def list_server_certificate_names
    response = @iam_client.list_server_certificates

    if response.server_certificate_metadata_list.empty?
      @logger.info('No server certificates found.')
      return
    end

    response.server_certificate_metadata_list.each do |certificate_metadata|
      @logger.info("Certificate Name:
#{certificate_metadata.server_certificate_name}")
    end
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing server certificates: #{e.message}")
  end

  # Updates the name of a server certificate.
  def update_server_certificate_name(current_name, new_name)
    @iam_client.update_server_certificate(
      server_certificate_name: current_name,
      new_server_certificate_name: new_name
    )
    @logger.info("Server certificate name updated from '#{current_name}' to
'#{new_name}'.")
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error updating server certificate name: #{e.message}")
    false
  end

  # Deletes a server certificate.
  def delete_server_certificate(name)
    @iam_client.delete_server_certificate(server_certificate_name: name)
    @logger.info("Server certificate '#{name}' deleted.")
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting server certificate: #{e.message}")
```

```
    false
  end
end
```

- For API details, see ListServerCertificates in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListSigningCertificates` with a CLI

The following code examples show how to use `ListSigningCertificates`.

CLI

AWS CLI

To list the signing certificates for an IAM user

The following `list-signing-certificates` example lists the signing certificates for the IAM user named Bob.

```
aws iam list-signing-certificates \
    --user-name Bob
```

Output:

```
{
    "Certificates": [
        {
            "UserName": "Bob",
            "Status": "Inactive",
            "CertificateBody": "-----BEGIN CERTIFICATE-----<certificate-
body>-----END CERTIFICATE-----",
            "CertificateId": "TA7SMP42TDN5Z26OBPJE7EXAMPLE",
            "UploadDate": "2013-06-06T21:40:08Z"
        }
    ]
}
```

For more information, see Manage signing certificates in the *Amazon EC2 User Guide*.

- For API details, see ListSigningCertificates in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves details about the signing certificate that is associated with the user named Bob.

```
Get-IAMSigningCertificate -UserName Bob
```

Output:

```
CertificateBody : -----BEGIN CERTIFICATE-----

 MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC

 VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6

 b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd

 BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN

 MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD

 VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z

 b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt
                  YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn
+a4GmWIWJ
                  21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/
f0wYK8m9T
                  rDHudUZg3qX4waLG5M43q7Wgc/
MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE

 Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
                  nUhVVxYUntneD9+h8Mg9q6q
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb

 FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
                  NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
```

```
                  -----END CERTIFICATE-----
CertificateId   : Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU
Status          : Active
UploadDate      : 4/20/2015 1:26:01 PM
UserName        : Bob
```

- For API details, see ListSigningCertificates in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves details about the signing certificate that is associated with the user named Bob.

```
Get-IAMSigningCertificate -UserName Bob
```

Output:

```
CertificateBody : -----BEGIN CERTIFICATE-----

 MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC

 VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6

 b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd

 BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN

 MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD

 VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z

 b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt
                  YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn
+a4GmWIWJ
                  21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/
f0wYK8m9T
                  rDHudUZg3qX4waLG5M43q7Wgc/
MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE

 Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
                  nUhVVxYUntneD9+h8Mg9q6q
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
```

```
 FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
                  NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
                  -----END CERTIFICATE-----
CertificateId   : Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU
Status          : Active
UploadDate      : 4/20/2015 1:26:01 PM
UserName        : Bob
```

- For API details, see ListSigningCertificates in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListUserPolicies` with an AWS SDK or CLI

The following code examples show how to use `ListUserPolicies`.

CLI

AWS CLI

To list policies for an IAM user

The following `list-user-policies` example lists the policies that are attached to the IAM user named Bob.

```
aws iam list-user-policies \
    --user-name Bob
```

Output:

```
{
    "PolicyNames": [
        "ExamplePolicy",
        "TestPolicy"
    ]
}
```

For more information, see Creating an IAM user in your AWS account in the *AWS IAM User Guide*.

- For API details, see ListUserPolicies in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(ctx context.Context, userName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListUserPolicies(ctx,
 &iam.ListUserPoliciesInput{
```

```
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName,
 err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}
```

- For API details, see ListUserPolicies in *AWS SDK for Go API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves the list of names of the inline policies that are embedded in the IAM user named `David`.

```
Get-IAMUserPolicyList -UserName David
```

Output:

```
Davids_IAM_Admin_Policy
```

- For API details, see ListUserPolicies in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves the list of names of the inline policies that are embedded in the IAM user named `David`.

```
Get-IAMUserPolicyList -UserName David
```

Output:

```
Davids_IAM_Admin_Policy
```

- For API details, see ListUserPolicies in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListUserTags` with a CLI

The following code examples show how to use `ListUserTags`.

CLI

AWS CLI

To list the tags attached to a user

The following `list-user-tags` example retrieves the list of tags associated with the specified IAM user.

```
aws iam list-user-tags \
    --user-name alice
```

Output:

```
{
    "Tags": [
        {
            "Key": "Department",
            "Value": "Accounting"
        },
        {
            "Key": "DeptID",
            "Value": "12345"
        }
    ],
    "IsTruncated": false
}
```

For more information, see Tagging IAM resources in the *AWS IAM User Guide*.

- For API details, see ListUserTags in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example fetches the tag associated with the user.

```
Get-IAMUserTagList -UserName joe
```

- For API details, see ListUserTags in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example fetches the tag associated with the user.

```
Get-IAMUserTagList -UserName joe
```

- For API details, see ListUserTags in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListUsers` with an AWS SDK or CLI

The following code examples show how to use `ListUsers`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create read-only and read-write users

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }
```

- For API details, see ListUsers in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}
```

```
###############################################################################
# function iam_list_users
#
# List the IAM users in the account.
#
# Returns:
#       The list of users names
#    And:
#       0 - If the user already exists.
#       1 - If the user doesn't exist.
###############################################################################
function iam_list_users() {
  local option OPTARG # Required to use getopts command in a function.
  local error_code
  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_list_users"
    echo "Lists the AWS Identity and Access Management (IAM) user in the
 account."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "h" option; do
    case "${option}" in
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  local response

  response=$(aws iam list-users \
    --output text \
    --query "Users[].UserName")
  error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports list-users operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- For API details, see ListUsers in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::listUsers(const Aws::Client::ClientConfiguration &clientConfig)
 {
    const Aws::String DATE_FORMAT = "%Y-%m-%d";
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::ListUsersRequest request;

    bool done = false;
    bool header = false;
    while (!done) {
        auto outcome = iam.ListUsers(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to list iam users:" <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        if (!header) {
```

```
            std::cout << std::left << std::setw(32) << "Name" <<
                      std::setw(30) << "ID" << std::setw(64) << "Arn" <<
                      std::setw(20) << "CreateDate" << std::endl;
            header = true;
        }

        const auto &users = outcome.GetResult().GetUsers();
        for (const auto &user: users) {
            std::cout << std::left << std::setw(32) << user.GetUserName() <<
                      std::setw(30) << user.GetUserId() << std::setw(64) <<
                      user.GetArn() << std::setw(20) <<
                      user.GetCreateDate().ToGmtString(DATE_FORMAT.c_str())
                      << std::endl;
        }

        if (outcome.GetResult().GetIsTruncated()) {
            request.SetMarker(outcome.GetResult().GetMarker());
        }
        else {
            done = true;
        }
    }

    return true;
}
```

- For API details, see ListUsers in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list IAM users

The following `list-users` example lists the IAM users in the current account.

```
aws iam list-users
```

Output:

```
{
    "Users": [
```

```
        {
            "UserName": "Adele",
            "Path": "/",
            "CreateDate": "2013-03-07T05:14:48Z",
            "UserId": "AKIAI44QH8DHBEXAMPLE",
            "Arn": "arn:aws:iam::123456789012:user/Adele"
        },
        {
            "UserName": "Bob",
            "Path": "/",
            "CreateDate": "2012-09-21T23:03:13Z",
            "UserId": "AKIAIOSFODNN7EXAMPLE",
            "Arn": "arn:aws:iam::123456789012:user/Bob"
        }
    ]
}
```

For more information, see Listing IAM users in the *AWS IAM User Guide*.

- For API details, see ListUsers in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
```

```
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(ctx context.Context, maxUsers int32)
 ([]types.User, error) {
 var users []types.User
 result, err := wrapper.IamClient.ListUsers(ctx, &iam.ListUsersInput{
  MaxItems: aws.Int32(maxUsers),
 })
 if err != nil {
  log.Printf("Couldn't list users. Here's why: %v\n", err)
 } else {
  users = result.Users
 }
 return users, err
}
```

- For API details, see ListUsers in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.AttachedPermissionsBoundary;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.ListUsersRequest;
```

```
import software.amazon.awssdk.services.iam.model.ListUsersResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.User;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListUsers {
    public static void main(String[] args) {
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        listAllUsers(iam);
        System.out.println("Done");
        iam.close();
    }

    public static void listAllUsers(IamClient iam) {
        try {
            boolean done = false;
            String newMarker = null;
            while (!done) {
                ListUsersResponse response;
                if (newMarker == null) {
                    ListUsersRequest request =
 ListUsersRequest.builder().build();
                    response = iam.listUsers(request);
                } else {
                    ListUsersRequest request = ListUsersRequest.builder()
                            .marker(newMarker)
                            .build();

                    response = iam.listUsers(request);
                }
```

```
                for (User user : response.users()) {
                    System.out.format("\n Retrieved user %s", user.userName());
                    AttachedPermissionsBoundary permissionsBoundary =
 user.permissionsBoundary();
                    if (permissionsBoundary != null)
                        System.out.format("\n Permissions boundary details %s",

 permissionsBoundary.permissionsBoundaryTypeAsString());
                }

                if (!response.isTruncated()) {
                    done = true;
                } else {
                    newMarker = response.marker();
                }
            }

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListUsers in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the users.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);

  for (const { UserName, CreateDate } of response.Users) {
    console.log(`${UserName} created on: ${CreateDate}`);
  }
  return response;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListUsers in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
```

```
      console.log("User " + user.UserName + " created", user.CreateDate);
    });
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListUsers in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listAllUsers() {
    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        val response = iamClient.listUsers(ListUsersRequest { })
        response.users?.forEach { user ->
            println("Retrieved user ${user.userName}")
            val permissionsBoundary = user.permissionsBoundary
            if (permissionsBoundary != null) {
                println("Permissions boundary details
 ${permissionsBoundary.permissionsBoundaryType}")
            }
        }
    }
}
```

- For API details, see ListUsers in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
$uuid = uniqid();
$service = new IAMService();

    public function listUsers($pathPrefix = "", $marker = "", $maxItems = 0)
    {
        $listUsersArguments = [];
        if ($pathPrefix) {
            $listUsersArguments["PathPrefix"] = $pathPrefix;
        }
        if ($marker) {
            $listUsersArguments["Marker"] = $marker;
        }
        if ($maxItems) {
            $listUsersArguments["MaxItems"] = $maxItems;
        }

        return $this->iamClient->listUsers($listUsersArguments);
    }
```

- For API details, see ListUsers in *AWS SDK for PHP API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves a collection of users in the current AWS account.

```
Get-IAMUserList
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Administrator
CreateDate       : 10/16/2014 9:03:09 AM
PasswordLastUsed : 3/4/2015 12:12:33 PM
Path             : /
UserId           : 7K3GJEANSKZF2EXAMPLE1
UserName         : Administrator

Arn              : arn:aws:iam::123456789012:user/Bob
CreateDate       : 4/6/2015 12:54:42 PM
PasswordLastUsed : 1/1/0001 12:00:00 AM
Path             : /
UserId           : L3EWNONDOM3YUEXAMPLE2
UserName         : bab

Arn              : arn:aws:iam::123456789012:user/David
CreateDate       : 12/10/2014 3:39:27 PM
PasswordLastUsed : 3/19/2015 8:44:04 AM
Path             : /
UserId           : Y4FKWQCXTA52QEXAMPLE3
UserName         : David
```

- For API details, see ListUsers in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves a collection of users in the current AWS account.

```
Get-IAMUserList
```

Output:

```
Arn              : arn:aws:iam::123456789012:user/Administrator
CreateDate       : 10/16/2014 9:03:09 AM
PasswordLastUsed : 3/4/2015 12:12:33 PM
Path             : /
UserId           : 7K3GJEANSKZF2EXAMPLE1
UserName         : Administrator

Arn              : arn:aws:iam::123456789012:user/Bob
CreateDate       : 4/6/2015 12:54:42 PM
PasswordLastUsed : 1/1/0001 12:00:00 AM
Path             : /
UserId           : L3EWNONDOM3YUEXAMPLE2
```

```
    UserName         : bab

    Arn              : arn:aws:iam::123456789012:user/David
    CreateDate       : 12/10/2014 3:39:27 PM
    PasswordLastUsed : 3/19/2015 8:44:04 AM
    Path             : /
    UserId           : Y4FKWQCXTA52QEXAMPLE3
    UserName         : David
```

- For API details, see ListUsers in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_users():
    """
    Lists the users in the current account.

    :return: The list of users.
    """
    try:
        users = list(iam.users.all())
        logger.info("Got %s users.", len(users))
    except ClientError:
        logger.exception("Couldn't get users.")
        raise
    else:
        return users
```

- For API details, see ListUsers in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Lists all users in the AWS account
#
# @return [Array<Aws::IAM::Types::User>] An array of user objects
def list_users
  users = []
  @iam_client.list_users.each_page do |page|
    page.users.each do |user|
      users << user
    end
  end
  users
rescue Aws::IAM::Errors::ServiceError => e
  @logger.error("Error listing users: #{e.message}")
  []
end
```

- For API details, see ListUsers in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
pub async fn list_users(
```

```
    client: &iamClient,
    path_prefix: Option<String>,
    marker: Option<String>,
    max_items: Option<i32>,
) -> Result<ListUsersOutput, SdkError<ListUsersError>> {
    let response = client
        .list_users()
        .set_path_prefix(path_prefix)
        .set_marker(marker)
        .set_max_items(max_items)
        .send()
        .await?;
    Ok(response)
}
```

- For API details, see ListUsers in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_iam->listusers( ).
        MESSAGE 'Retrieved user list.' TYPE 'I'.
      CATCH /aws1/cx_iamservicefailureex.
        MESSAGE 'Service failure when listing users.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListUsers in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    public func listUsers() async throws -> [MyUserRecord] {
        var userList: [MyUserRecord] = []

        // Use "Paginated" to get all the users.
        // This lets the SDK handle the 'isTruncated' in "ListUsersOutput".
        let input = ListUsersInput()
        let output = client.listUsersPaginated(input: input)

        do {
            for try await page in output {
                guard let users = page.users else {
                    continue
                }
                for user in users {
                    if let id = user.userId, let name = user.userName {
                        userList.append(MyUserRecord(id: id, name: name))
                    }
                }
            }
        }
        catch {
            print("ERROR: listUsers:", dump(error))
            throw error
        }
       return userList
    }
```

- For API details, see ListUsers in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListVirtualMfaDevices` with a CLI

The following code examples show how to use `ListVirtualMfaDevices`.

CLI

AWS CLI

To list virtual MFA devices

The following `list-virtual-mfa-devices` example lists the virtual MFA devices that have been configured for the current account.

```
aws iam list-virtual-mfa-devices
```

Output:

```
{
    "VirtualMFADevices": [
        {
            "SerialNumber": "arn:aws:iam::123456789012:mfa/ExampleMFADevice"
        },
        {
            "SerialNumber": "arn:aws:iam::123456789012:mfa/Fred"
        }
    ]
}
```

For more information, see Enabling a virtual multi-factor authentication (MFA) device in the *AWS IAM User Guide*.

- For API details, see ListVirtualMfaDevices in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example retrieves a collection of the virtual MFA devices that are assigned to users in the AWS account. The `User` property of each is an object with details of the IAM user to which the device is assigned.

```
Get-IAMVirtualMFADevice -AssignmentStatus Assigned
```

Output:

```
Base32StringSeed :
EnableDate       : 4/13/2015 12:03:42 PM
QRCodePNG        :
SerialNumber     : arn:aws:iam::123456789012:mfa/David
User             : Amazon.IdentityManagement.Model.User

Base32StringSeed :
EnableDate       : 4/13/2015 12:06:41 PM
QRCodePNG        :
SerialNumber     : arn:aws:iam::123456789012:mfa/root-account-mfa-device
User             : Amazon.IdentityManagement.Model.User
```

- For API details, see ListVirtualMfaDevices in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example retrieves a collection of the virtual MFA devices that are assigned to users in the AWS account. The `User` property of each is an object with details of the IAM user to which the device is assigned.

```
Get-IAMVirtualMFADevice -AssignmentStatus Assigned
```

Output:

```
Base32StringSeed :
EnableDate       : 4/13/2015 12:03:42 PM
QRCodePNG        :
SerialNumber     : arn:aws:iam::123456789012:mfa/David
```

```
User             : Amazon.IdentityManagement.Model.User

Base32StringSeed :
EnableDate       : 4/13/2015 12:06:41 PM
QRCodePNG        :
SerialNumber     : arn:aws:iam::123456789012:mfa/root-account-mfa-device
User             : Amazon.IdentityManagement.Model.User
```

- For API details, see ListVirtualMfaDevices in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutGroupPolicy` with a CLI

The following code examples show how to use `PutGroupPolicy`.

CLI

AWS CLI

To add a policy to a group

The following `put-group-policy` example adds a policy to the IAM group named `Admins`.

```
aws iam put-group-policy \
    --group-name Admins \
    --policy-document file://AdminPolicy.json \
    --policy-name AdminRoot
```

Contents of `AdminPolicy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
```

```
        }
    ]
}
```

This command produces no output.

The policy is defined as a JSON document in the *AdminPolicy.json* file. (The file name and extension do not have significance.)

For more information, see Managing IAM policies in the *AWS IAM User Guide*.

- For API details, see PutGroupPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates an inline policy named `AppTesterPolicy` and embeds it in the IAM group `AppTesters`. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes the file `apptesterpolicy.json`. Note that you must use the `-Raw` parameter to successfully process the content of the JSON file.

```
Write-IAMGroupPolicy -GroupName AppTesters -PolicyName AppTesterPolicy -
PolicyDocument (Get-Content -Raw apptesterpolicy.json)
```

- For API details, see PutGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates an inline policy named `AppTesterPolicy` and embeds it in the IAM group `AppTesters`. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes the file `apptesterpolicy.json`. Note that you must use the `-Raw` parameter to successfully process the content of the JSON file.

```
Write-IAMGroupPolicy -GroupName AppTesters -PolicyName AppTesterPolicy -
PolicyDocument (Get-Content -Raw apptesterpolicy.json)
```

- For API details, see PutGroupPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutRolePermissionsBoundary` with a CLI

The following code examples show how to use `PutRolePermissionsBoundary`.

CLI

AWS CLI

Example 1: To apply a permissions boundary based on a custom policy to an IAM role

The following `put-role-permissions-boundary` example applies the custom policy named `intern-boundary` as the permissions boundary for the specified IAM role.

```
aws iam put-role-permissions-boundary \
    --permissions-boundary arn:aws:iam::123456789012:policy/intern-boundary \
    --role-name lambda-application-role
```

This command produces no output.

Example 2: To apply a permissions boundary based on an AWS managed policy to an IAM role

The following `put-role-permissions-boundary` example applies the AWS managed `PowerUserAccess` policy as the permissions boundary for the specified IAM role.

```
aws iam put-role-permissions-boundary \
    --permissions-boundary arn:aws:iam::aws:policy/PowerUserAccess \
    --role-name x-account-admin
```

This command produces no output.

For more information, see IAM role management in the *AWS IAM User Guide*.

- For API details, see PutRolePermissionsBoundary in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example shows how to set the Permission boundary for a IAM Role. You can set AWS Managed policies or Custom policies as permission boundary.

```
Set-IAMRolePermissionsBoundary -RoleName MyRoleName -PermissionsBoundary
 arn:aws:iam::123456789012:policy/intern-boundary
```

- For API details, see PutRolePermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example shows how to set the Permission boundary for a IAM Role. You can set AWS Managed policies or Custom policies as permission boundary.

```
Set-IAMRolePermissionsBoundary -RoleName MyRoleName -PermissionsBoundary
 arn:aws:iam::123456789012:policy/intern-boundary
```

- For API details, see PutRolePermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutRolePolicy` with an AWS SDK or CLI

The following code examples show how to use `PutRolePolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Get started with serverless data warehouses
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with provisioned data warehouse clusters

- Run CPU stress tests on virtual machine instances using fault injection

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string
 roleName, string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see PutRolePolicy in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::putRolePolicy(
        const Aws::String &roleName,
        const Aws::String &policyName,
        const Aws::String &policyDocument,
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::IAM::IAMClient iamClient(clientConfig);
    Aws::IAM::Model::PutRolePolicyRequest request;

    request.SetRoleName(roleName);
    request.SetPolicyName(policyName);
    request.SetPolicyDocument(policyDocument);

    Aws::IAM::Model::PutRolePolicyOutcome outcome =
 iamClient.PutRolePolicy(request);
    if (!outcome.IsSuccess()) {
        std::cerr << "Error putting policy on role. " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Successfully put the role policy." << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see PutRolePolicy in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To attach a permissions policy to an IAM role

The following `put-role-policy` example adds a permissions policy to the role named `Test-Role`.

```
aws iam put-role-policy \
    --role-name Test-Role \
    --policy-name ExamplePolicy \
    --policy-document file://AdminPolicy.json
```

Contents of `AdminPolicy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

This command produces no output.

The policy is defined as a JSON document in the *AdminPolicy.json* file. (The file name and extension do not have significance.)

To attach a trust policy to a role, use the `update-assume-role-policy` command.

For more information, see IAM role management in the *AWS IAM User Guide*.

- For API details, see PutRolePolicy in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::amzn-s3-demo-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- For API details, see PutRolePolicy in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates an inline policy named `FedTesterRolePolicy` and embeds it in the IAM role `FedTesterRole`. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes from the file `FedTesterPolicy.json`. Note that you must use the `-Raw` parameter to successfully process the content of the JSON file.

```
Write-IAMRolePolicy -RoleName FedTesterRole -PolicyName FedTesterRolePolicy -
PolicyDocument (Get-Content -Raw FedTesterPolicy.json)
```

- For API details, see PutRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates an inline policy named `FedTesterRolePolicy` and embeds it in the IAM role `FedTesterRole`. If an inline policy with the same name

already exists, then it is overwritten. The JSON policy content comes from the file `FedTesterPolicy.json`. Note that you must use the `-Raw` parameter to successfully process the content of the JSON file.

```
Write-IAMRolePolicy -RoleName FedTesterRole -PolicyName FedTesterRolePolicy -
PolicyDocument (Get-Content -Raw FedTesterPolicy.json)
```

- For API details, see PutRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutUserPermissionsBoundary` with a CLI

The following code examples show how to use `PutUserPermissionsBoundary`.

CLI

AWS CLI

Example 1: To apply a permissions boundary based on a custom policy to an IAM user

The following `put-user-permissions-boundary` example applies a custom policy named `intern-boundary` as the permissions boundary for the specified IAM user.

```
aws iam put-user-permissions-boundary \
    --permissions-boundary arn:aws:iam::123456789012:policy/intern-boundary \
    --user-name intern
```

This command produces no output.

Example 2: To apply a permissions boundary based on an AWS managed policy to an IAM user

The following `put-user-permissions-boundary` example applies the AWS managed pollicy named `PowerUserAccess` as the permissions boundary for the specified IAM user.

```
aws iam put-user-permissions-boundary \
    --permissions-boundary arn:aws:iam::aws:policy/PowerUserAccess \
```

```
    --user-name developer
```

This command produces no output.

For more information, see Adding and removing IAM identity permissions in the *AWS IAM User Guide*.

- For API details, see PutUserPermissionsBoundary in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example shows how to set the Permission boundary for the user. You can set AWS Managed policies or Custom policies as permission boundary.

```
Set-IAMUserPermissionsBoundary -UserName joe -PermissionsBoundary
 arn:aws:iam::123456789012:policy/intern-boundary
```

- For API details, see PutUserPermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example shows how to set the Permission boundary for the user. You can set AWS Managed policies or Custom policies as permission boundary.

```
Set-IAMUserPermissionsBoundary -UserName joe -PermissionsBoundary
 arn:aws:iam::123456789012:policy/intern-boundary
```

- For API details, see PutUserPermissionsBoundary in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutUserPolicy` with an AWS SDK or CLI

The following code examples show how to use `PutUserPolicy`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

CLI

AWS CLI

To attach a policy to an IAM user

The following `put-user-policy` example attaches a policy to the IAM user named Bob.

```
aws iam put-user-policy \
    --user-name Bob \
    --policy-name ExamplePolicy \
    --policy-document file://AdminPolicy.json
```

Contents of `AdminPolicy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

This command produces no output.

The policy is defined as a JSON document in the *AdminPolicy.json* file. (The file name and extension do not have significance.)

For more information, see Adding and removing IAM identity permissions in the *AWS IAM User Guide*.

- For API details, see PutUserPolicy in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// CreateUserPolicy adds an inline policy to a user. This example creates a
 policy that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure
 and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(ctx context.Context, userName string,
 policyName string, actions []string,
 roleArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
```

```
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(roleArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
 err)
  return err
 }
 _, err = wrapper.IamClient.PutUserPolicy(ctx, &iam.PutUserPolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
  UserName:       aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName,
 err)
 }
 return err
}
```

- For API details, see PutUserPolicy in *AWS SDK for Go API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates an inline policy named `EC2AccessPolicy` and embeds it in the IAM user Bob. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes from the file `EC2AccessPolicy.json`. Note that you must use the `-Raw` parameter to successfully process the content of the JSON file.

```
Write-IAMUserPolicy -UserName Bob -PolicyName EC2AccessPolicy -PolicyDocument
 (Get-Content -Raw EC2AccessPolicy.json)
```

- For API details, see PutUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates an inline policy named EC2AccessPolicy and embeds it in the IAM user Bob. If an inline policy with the same name already exists, then it is overwritten. The JSON policy content comes from the file EC2AccessPolicy.json. Note that you must use the -Raw parameter to successfully process the content of the JSON file.

```
Write-IAMUserPolicy -UserName Bob -PolicyName EC2AccessPolicy -PolicyDocument
 (Get-Content -Raw EC2AccessPolicy.json)
```

- For API details, see PutUserPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
  # Creates an inline policy for a specified user.
  # @param username [String] The name of the IAM user.
  # @param policy_name [String] The name of the policy to create.
  # @param policy_document [String] The JSON policy document.
  # @return [Boolean]
  def create_user_policy(username, policy_name, policy_document)
    @iam_client.put_user_policy({
                                  user_name: username,
                                  policy_name: policy_name,
                                  policy_document: policy_document
                                })
    @logger.info("Policy #{policy_name} created for user #{username}.")
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Couldn't create policy #{policy_name} for user #{username}.
Here's why:")
    @logger.error("\t#{e.code}: #{e.message}")
```

```
    false
  end
```

- For API details, see PutUserPolicy in *AWS SDK for Ruby API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSIAM
import AWSS3


    func putUserPolicy(policyDocument: String, policyName: String, user:
 IAMClientTypes.User) async throws {
        let input = PutUserPolicyInput(
            policyDocument: policyDocument,
            policyName: policyName,
            userName: user.userName
        )
        do {
            _ = try await iamClient.putUserPolicy(input: input)
        } catch {
            print("ERROR: putUserPolicy:", dump(error))
            throw error
        }
    }
```

- For API details, see PutUserPolicy in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RemoveClientIdFromOpenIdConnectProvider` with a CLI

The following code examples show how to use `RemoveClientIdFromOpenIdConnectProvider`.

CLI

AWS CLI

To remove the specified client ID from the list of client IDs registered for the specified IAM OpenID Connect provider

The following `remove-client-id-from-open-id-connect-provider` example removes the client ID `My-TestApp-3` from the list of client IDs associated with the IAM OIDC provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com`.

```
aws iam remove-client-id-from-open-id-connect-provider \
    --client-id My-TestApp-3 \
    --open-id-connect-provider-arn arn:aws:iam::123456789012:oidc-provider/
example.oidcprovider.com
```

This command produces no output.

For more information, see Creating OpenID Connect (OIDC) identity providers in the *AWS IAM User Guide*.

- For API details, see RemoveClientIdFromOpenIdConnectProvider in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example removes the client ID `My-TestApp-3` from the list of client IDs associated with the IAM OIDC provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com`.

```
Remove-IAMClientIDFromOpenIDConnectProvider -ClientID My-TestApp-3
 -OpenIDConnectProviderArn arn:aws:iam::123456789012:oidc-provider/
example.oidcprovider.com
```

- For API details, see RemoveClientIdFromOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example removes the client ID `My-TestApp-3` from the list of client IDs associated with the IAM OIDC provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com`.

```
Remove-IAMClientIDFromOpenIDConnectProvider -ClientID My-TestApp-3
 -OpenIDConnectProviderArn arn:aws:iam::123456789012:oidc-provider/
example.oidcprovider.com
```

- For API details, see RemoveClientIdFromOpenIdConnectProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RemoveRoleFromInstanceProfile` with a CLI

The following code examples show how to use `RemoveRoleFromInstanceProfile`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Getting started with managed streaming
- Run CPU stress tests on virtual machine instances using fault injection

CLI

AWS CLI

To remove a role from an instance profile

The following `remove-role-from-instance-profile` example removes the role named `Test-Role` from the instance profile named `ExampleInstanceProfile`.

```
aws iam remove-role-from-instance-profile \
    --instance-profile-name ExampleInstanceProfile \
    --role-name Test-Role
```

This command produces no output.

For more information, see Using instance profiles in the *AWS IAM User Guide*.

- For API details, see RemoveRoleFromInstanceProfile in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example deletes the role named `MyNewRole` from the EC2 instance profile named `MyNewRole`. An instance profile that is created in the IAM console always has the same name as the role, as in this example. If you create them in the API or CLI, then they can have different names.

```
Remove-IAMRoleFromInstanceProfile -InstanceProfileName MyNewRole -RoleName
 MyNewRole -Force
```

- For API details, see RemoveRoleFromInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example deletes the role named `MyNewRole` from the EC2 instance profile named `MyNewRole`. An instance profile that is created in the IAM console always has the same name as the role, as in this example. If you create them in the API or CLI, then they can have different names.

```
Remove-IAMRoleFromInstanceProfile -InstanceProfileName MyNewRole -RoleName
 MyNewRole -Force
```

- For API details, see RemoveRoleFromInstanceProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RemoveUserFromGroup` with a CLI

The following code examples show how to use `RemoveUserFromGroup`.

CLI

AWS CLI

To remove a user from an IAM group

The following `remove-user-from-group` example removes the user named Bob from the IAM group named `Admins`.

```
aws iam remove-user-from-group \
    --user-name Bob \
    --group-name Admins
```

This command produces no output.

For more information, see Adding and removing users in an IAM user group in the *AWS IAM User Guide*.

- For API details, see RemoveUserFromGroup in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example removes the IAM user Bob from the group `Testers`.

```
Remove-IAMUserFromGroup -GroupName Testers -UserName Bob
```

Example 2: This example finds any groups of which IAM user `Theresa` is a member, and then removes `Theresa` from those groups.

```
$groups = Get-IAMGroupForUser -UserName Theresa
foreach ($group in $groups) { Remove-IAMUserFromGroup -GroupName $group.GroupName
 -UserName Theresa -Force }
```

Example 3: This example shows an alternate way of removing the IAM user Bob from the `Testers` group.

```
Get-IAMGroupForUser -UserName Bob | Remove-IAMUserFromGroup -UserName Bob -
GroupName Testers -Force
```

- For API details, see RemoveUserFromGroup in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example removes the IAM user Bob from the group `Testers`.

```
Remove-IAMUserFromGroup -GroupName Testers -UserName Bob
```

Example 2: This example finds any groups of which IAM user `Theresa` is a member, and then removes `Theresa` from those groups.

```
$groups = Get-IAMGroupForUser -UserName Theresa
foreach ($group in $groups) { Remove-IAMUserFromGroup -GroupName $group.GroupName
 -UserName Theresa -Force }
```

Example 3: This example shows an alternate way of removing the IAM user Bob from the `Testers` group.

```
Get-IAMGroupForUser -UserName Bob | Remove-IAMUserFromGroup -UserName Bob -
GroupName Testers -Force
```

- For API details, see RemoveUserFromGroup in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ResyncMfaDevice` with a CLI

The following code examples show how to use `ResyncMfaDevice`.

CLI

AWS CLI

To synchronize an MFA device

The following `resync-mfa-device` example synchronizes the MFA device that is associated with the IAM user Bob and whose ARN is `arn:aws:iam::123456789012:mfa/BobsMFADevice` with an authenticator program that provided the two authentication codes.

```
aws iam resync-mfa-device \
    --user-name Bob \
    --serial-number arn:aws:iam::123456789012:mfa/BobsMFADevice \
    --authentication-code1 123456 \
    --authentication-code2 987654
```

This command produces no output.

For more information, see Using multi-factor authentication (MFA) in AWS in the *AWS IAM User Guide*.

- For API details, see ResyncMfaDevice in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example synchronizes the MFA device that is associated with the IAM user Bob and whose ARN is `arn:aws:iam::123456789012:mfa/bob` with an authenticator program that provided the two authentication codes.

```
Sync-IAMMFADevice -SerialNumber arn:aws:iam::123456789012:mfa/theresa -
AuthenticationCode1 123456 -AuthenticationCode2 987654 -UserName Bob
```

Example 2: This example synchronizes the IAM MFA device that is associated with the IAM user `Theresa` with a physical device that has the serial number `ABCD12345678` and that provided the two authentication codes.

```
Sync-IAMMFADevice -SerialNumber ABCD12345678 -AuthenticationCode1 123456 -
AuthenticationCode2 987654 -UserName Theresa
```

- For API details, see ResyncMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example synchronizes the MFA device that is associated with the IAM user Bob and whose ARN is `arn:aws:iam::123456789012:mfa/bob` with an authenticator program that provided the two authentication codes.

```
Sync-IAMMFADevice -SerialNumber arn:aws:iam::123456789012:mfa/theresa -
AuthenticationCode1 123456 -AuthenticationCode2 987654 -UserName Bob
```

Example 2: This example synchronizes the IAM MFA device that is associated with the IAM user `Theresa` with a physical device that has the serial number `ABCD12345678` and that provided the two authentication codes.

```
Sync-IAMMFADevice -SerialNumber ABCD12345678 -AuthenticationCode1 123456 -
AuthenticationCode2 987654 -UserName Theresa
```

- For API details, see ResyncMfaDevice in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetDefaultPolicyVersion` with an AWS SDK or CLI

The following code examples show how to use `SetDefaultPolicyVersion`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Manage policies
- Roll back a policy version

CLI

AWS CLI

To set the specified version of the specified policy as the policy's default version.

The following `set-default-policy-version` example sets the `v2` version of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` as the default active version.

```
aws iam set-default-policy-version \
    --policy-arn arn:aws:iam::123456789012:policy/MyPolicy \
    --version-id v2
```

This command produces no output.

For more information, see Policies and permissions in IAM in the *AWS IAM User Guide*.

- For API details, see SetDefaultPolicyVersion in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example sets the v2 version of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` as the default active version.

```
Set-IAMDefaultPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/MyPolicy
 -VersionId v2
```

- For API details, see SetDefaultPolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example sets the v2 version of the policy whose ARN is `arn:aws:iam::123456789012:policy/MyPolicy` as the default active version.

```
Set-IAMDefaultPolicyVersion -PolicyArn arn:aws:iam::123456789012:policy/MyPolicy
 -VersionId v2
```

- For API details, see SetDefaultPolicyVersion in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_iam->setdefaultpolicyversion(
      iv_policyarn = iv_policy_arn
      iv_versionid = iv_version_id ).
    MESSAGE 'Default policy version set successfully.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'Policy or version does not exist.' TYPE 'E'.
  CATCH /aws1/cx_iaminvalidinputex.
    MESSAGE 'Invalid input provided.' TYPE 'E'.
  CATCH /aws1/cx_iamlimitexceededex.
    MESSAGE 'Limit exceeded.' TYPE 'E'.
ENDTRY.
```

- For API details, see SetDefaultPolicyVersion in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `TagRole` with a CLI

The following code examples show how to use `TagRole`.

CLI

AWS CLI

To add a tag to a role

The following `tag-role` example adds a tag with a Department name to the specified role.

```
aws iam tag-role \
    --role-name my-role \
    --tags '{"Key": "Department", "Value": "Accounting"}'
```

This command produces no output.

For more information, see Tagging IAM resources in the *AWS IAM User Guide*.

- For API details, see TagRole in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example adds tag to Role in Identity Management Service

```
Add-IAMRoleTag -RoleName AdminRoleacess -Tag @{ Key = 'abac'; Value = 'testing'}
```

- For API details, see TagRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example adds tag to Role in Identity Management Service

```
Add-IAMRoleTag -RoleName AdminRoleacess -Tag @{ Key = 'abac'; Value = 'testing'}
```

- For API details, see TagRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `TagUser` with a CLI

The following code examples show how to use `TagUser`.

CLI

AWS CLI

To add a tag to a user

The following `tag-user` example adds a tag with the associated Department to the specified user.

```
aws iam tag-user \
    --user-name alice \
    --tags '{"Key": "Department", "Value": "Accounting"}'
```

This command produces no output.

For more information, see Tagging IAM resources in the *AWS IAM User Guide*.

- For API details, see TagUser in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example adds tag to User in Identity Management Service

```
Add-IAMUserTag -UserName joe -Tag @{ Key = 'abac'; Value = 'testing'}
```

- For API details, see TagUser in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example adds tag to User in Identity Management Service

```
Add-IAMUserTag -UserName joe -Tag @{ Key = 'abac'; Value = 'testing'}
```

- For API details, see TagUser in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UntagRole` with a CLI

The following code examples show how to use `UntagRole`.

CLI

AWS CLI

To remove a tag from a role

The following `untag-role` example removes any tag with the key name 'Department' from the specified role.

```
aws iam untag-role \
    --role-name my-role \
    --tag-keys Department
```

This command produces no output.

For more information, see Tagging IAM resources in the *AWS IAM User Guide*.

- For API details, see UntagRole in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example removes the tag from the role named "MyRoleName" with tag key as "abac". To remove multiple tags, provide a comma separted tag keys list.

```
Remove-IAMRoleTag -RoleName MyRoleName -TagKey "abac","xyzw"
```

- For API details, see UntagRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example removes the tag from the role named "MyRoleName" with tag key as "abac". To remove multiple tags, provide a comma separted tag keys list.

```
Remove-IAMRoleTag -RoleName MyRoleName -TagKey "abac","xyzw"
```

- For API details, see UntagRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UntagUser` with a CLI

The following code examples show how to use `UntagUser`.

CLI

AWS CLI

To remove a tag from a user

The following `untag-user` example removes any tag with the key name 'Department' from the specified user.

```
aws iam untag-user \
    --user-name alice \
    --tag-keys Department
```

This command produces no output.

For more information, see Tagging IAM resources in the *AWS IAM User Guide*.

- For API details, see UntagUser in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example removes the tag from the user named "joe" with tag key as "abac" and "xyzw". To remove multiple tags, provide a comma separted tag keys list.

```
Remove-IAMUserTag -UserName joe -TagKey "abac","xyzw"
```

- For API details, see UntagUser in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example removes the tag from the user named "joe" with tag key as "abac" and "xyzw". To remove multiple tags, provide a comma separted tag keys list.

```
Remove-IAMUserTag -UserName joe -TagKey "abac","xyzw"
```

- For API details, see UntagUser in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateAccessKey` with an AWS SDK or CLI

The following code examples show how to use `UpdateAccessKey`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage access keys

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
##############################################################################
# function iam_update_access_key
#
# This function can activate or deactivate an IAM access key for the specified
 IAM user.
#
# Parameters:
#       -u user_name  -- The name of the user.
#       -k access_key -- The access key to update.
#       -a            -- Activate the selected access key.
#       -d            -- Deactivate the selected access key.
#
# Example:
#       # To deactivate the selected access key for IAM user Bob
#       iam_update_access_key -u Bob -k AKIAIOSFODNN7EXAMPLE -d
#
# Returns:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function iam_update_access_key() {
  local user_name access_key status response
  local option OPTARG # Required to use getopts command in a function.
  local activate_flag=false deactivate_flag=false

  # bashsupport disable=BP5008
  function usage() {
    echo "function iam_update_access_key"
    echo "Updates the status of an AWS Identity and Access Management (IAM)
 access key for the specified IAM user"
    echo "  -u user_name    The name of the user."
    echo "  -k access_key   The access key to update."
    echo "  -a              Activate the access key."
    echo "  -d              Deactivate the access key."
    echo ""
  }

  # Retrieve the calling parameters.
    while getopts "u:k:adh" option; do
      case "${option}" in
        u) user_name="${OPTARG}" ;;
        k) access_key="${OPTARG}" ;;
        a) activate_flag=true ;;
        d) deactivate_flag=true ;;
        h)
          usage
          return 0
          ;;
        \?)
          echo "Invalid parameter"
          usage
          return 1
          ;;
      esac
    done
    export OPTIND=1

   # Validate input parameters
    if [[ -z "$user_name" ]]; then
      errecho "ERROR: You must provide a username with the -u parameter."
      usage
      return 1
```

```
    fi

    if [[ -z "$access_key" ]]; then
      errecho "ERROR: You must provide an access key with the -k parameter."
      usage
      return 1
    fi

    # Ensure that only -a or -d is specified
    if [[ "$activate_flag" == true && "$deactivate_flag" == true ]]; then
      errecho "ERROR: You cannot specify both -a (activate) and -d (deactivate)
at the same time."
      usage
      return 1
    fi

    # If neither -a nor -d is provided, return an error
    if [[ "$activate_flag" == false && "$deactivate_flag" == false ]]; then
      errecho "ERROR: You must specify either -a (activate) or -d (deactivate)."
      usage
      return 1
    fi

    # Determine the status based on the flag
    if [[ "$activate_flag" == true ]]; then
      status="Active"
    elif [[ "$deactivate_flag" == true ]]; then
      status="Inactive"
    fi

    iecho "Parameters:\n"
    iecho "    Username:   $user_name"
    iecho "    Access key: $access_key"
    iecho "    New status: $status"
    iecho ""

    # Update the access key status
    response=$(aws iam update-access-key \
      --user-name "$user_name" \
      --access-key-id "$access_key" \
      --status "$status" 2>&1)

    local error_code=${?}
```

```
    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log $error_code
      errecho "ERROR: AWS reports update-access-key operation failed.\n$response"
      return 1
    fi

    iecho "update-access-key response: $response"
    iecho

    return 0
}
```

- For API details, see UpdateAccessKey in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::updateAccessKey(const Aws::String &userName,
                                  const Aws::String &accessKeyID,
                                  Aws::IAM::Model::StatusType status,
                                  const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::UpdateAccessKeyRequest request;
    request.SetUserName(userName);
    request.SetAccessKeyId(accessKeyID);
    request.SetStatus(status);

    auto outcome = iam.UpdateAccessKey(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated status of access key "
                  << accessKeyID << " for user " << userName << std::endl;
    }
    else {
```

```
        std::cerr << "Error updated status of access key " << accessKeyID <<
                  " for user " << userName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see UpdateAccessKey in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To activate or deactivate an access key for an IAM user

The following `update-access-key` example deactivates the specified access key (access key ID and secret access key) for the IAM user named Bob.

```
aws iam update-access-key \
    --access-key-id AKIAIOSFODNN7EXAMPLE \
    --status Inactive \
    --user-name Bob
```

This command produces no output.

Deactivating the key means that it cannot be used for programmatic access to AWS. However, the key is still available and can be reactivated.

For more information, see Managing access keys for IAM users in the *AWS IAM User Guide*.

- For API details, see UpdateAccessKey in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.StatusType;
import software.amazon.awssdk.services.iam.model.UpdateAccessKeyRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class UpdateAccessKey {

    private static StatusType statusType;

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <username> <accessId> <status>\s

                Where:
                    username - The name of the user whose key you want to update.
\s
                    accessId - The access key ID of the secret access key you
 want to update.\s
                    status - The status you want to assign to the secret access
 key.\s
```

```
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String username = args[0];
        String accessId = args[1];
        String status = args[2];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        updateKey(iam, username, accessId, status);
        System.out.println("Done");
        iam.close();
    }

    public static void updateKey(IamClient iam, String username, String accessId,
String status) {
        try {
            if (status.toLowerCase().equalsIgnoreCase("active")) {
                statusType = StatusType.ACTIVE;
            } else if (status.toLowerCase().equalsIgnoreCase("inactive")) {
                statusType = StatusType.INACTIVE;
            } else {
                statusType = StatusType.UNKNOWN_TO_SDK_VERSION;
            }

            UpdateAccessKeyRequest request = UpdateAccessKeyRequest.builder()
                    .accessKeyId(accessId)
                    .userName(username)
                    .status(statusType)
                    .build();

            iam.updateAccessKey(request);
            System.out.printf("Successfully updated the status of access key %s
to" +
                    "status %s for user %s", accessId, status, username);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
}
```

- For API details, see UpdateAccessKey in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Update the access key.

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see UpdateAccessKey in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see UpdateAccessKey in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example changes the status of the access key `AKIAIOSFODNN7EXAMPLE` for the IAM user named `Bob` to `Inactive`.

```
Update-IAMAccessKey -UserName Bob -AccessKeyId AKIAIOSFODNN7EXAMPLE -Status
 Inactive
```

- For API details, see UpdateAccessKey in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example changes the status of the access key `AKIAIOSFODNN7EXAMPLE` for the IAM user named `Bob` to `Inactive`.

```
Update-IAMAccessKey -UserName Bob -AccessKeyId AKIAIOSFODNN7EXAMPLE -Status
 Inactive
```

- For API details, see UpdateAccessKey in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def update_key(user_name, key_id, activate):
    """
    Updates the status of a key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to update.
    :param activate: When True, the key is activated. Otherwise, the key is
 deactivated.
    """
```

```
    try:
        key = iam.User(user_name).AccessKey(key_id)
        if activate:
            key.activate()
        else:
            key.deactivate()
        logger.info("%s key %s.", "Activated" if activate else "Deactivated",
key_id)
    except ClientError:
        logger.exception(
            "Couldn't %s key %s.", "Activate" if activate else "Deactivate",
key_id
        )
        raise
```

- For API details, see UpdateAccessKey in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_iam->updateaccesskey(
          iv_accesskeyid = iv_access_key_id
          iv_status = iv_status
          iv_username = iv_user_name ).
        MESSAGE 'Access key updated successfully.' TYPE 'I'.
      CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'Access key or user does not exist.' TYPE 'E'.
    ENDTRY.
```

- For API details, see UpdateAccessKey in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateAccountPasswordPolicy` with a CLI

The following code examples show how to use `UpdateAccountPasswordPolicy`.

CLI

AWS CLI

To set or change the current account password policy

The following `update-account-password-policy` example sets the password policy to require a minimum length of eight characters and to require one or more numbers in the password.

```
aws iam update-account-password-policy \
    --minimum-password-length 8 \
    --require-numbers
```

This command produces no output.

Changes to an account's password policy affect any new passwords that are created for IAM users in the account. Password policy changes do not affect existing passwords.

For more information, see Setting an account password policy for IAM users in the *AWS IAM User Guide*.

- For API details, see UpdateAccountPasswordPolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example updates the password policy for the account with the specified settings. Note that any parameters that are not included in the command are not left unmodified. Instead, they are reset to default values.

```
Update-IAMAccountPasswordPolicy -AllowUsersToChangePasswords $true -HardExpiry
 $false -MaxPasswordAge 90 -MinimumPasswordLength 8 -PasswordReusePrevention 20
 -RequireLowercaseCharacters $true -RequireNumbers $true -RequireSymbols $true -
RequireUppercaseCharacters $true
```

- For API details, see UpdateAccountPasswordPolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example updates the password policy for the account with the specified settings. Note that any parameters that are not included in the command are not left unmodified. Instead, they are reset to default values.

```
Update-IAMAccountPasswordPolicy -AllowUsersToChangePasswords $true -HardExpiry
 $false -MaxPasswordAge 90 -MinimumPasswordLength 8 -PasswordReusePrevention 20
 -RequireLowercaseCharacters $true -RequireNumbers $true -RequireSymbols $true -
RequireUppercaseCharacters $true
```

- For API details, see UpdateAccountPasswordPolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateAssumeRolePolicy` with a CLI

The following code examples show how to use `UpdateAssumeRolePolicy`.

CLI

AWS CLI

To update the trust policy for an IAM role

The following `update-assume-role-policy` example updates the trust policy for the role named `Test-Role`.

```
aws iam update-assume-role-policy \
```

```
    --role-name Test-Role \
    --policy-document file://Test-Role-Trust-Policy.json
```

Contents of `Test-Role-Trust-Policy.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

This command produces no output.

The trust policy is defined as a JSON document in the *Test-Role-Trust-Policy.json* file. (The file name and extension do not have significance.) The trust policy must specify a principal.

To update the permissions policy for a role, use the `put-role-policy` command.

For more information, see Creating IAM roles in the *AWS IAM User Guide*.

- For API details, see UpdateAssumeRolePolicy in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example updates the IAM role named `ClientRole` with a new trust policy, the contents of which come from the file `ClientRolePolicy.json`. Note that you must use the `-Raw` switch parameter to successfully process the contents of the JSON file.

```
Update-IAMAssumeRolePolicy -RoleName ClientRole -PolicyDocument (Get-Content -raw
 ClientRolePolicy.json)
```

- For API details, see UpdateAssumeRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example updates the IAM role named `ClientRole` with a new trust policy, the contents of which come from the file `ClientRolePolicy.json`. Note that you must use the `-Raw` switch parameter to successfully process the contents of the JSON file.

```
Update-IAMAssumeRolePolicy -RoleName ClientRole -PolicyDocument (Get-Content -raw
 ClientRolePolicy.json)
```

- For API details, see UpdateAssumeRolePolicy in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateGroup` with a CLI

The following code examples show how to use `UpdateGroup`.

CLI

AWS CLI

To rename an IAM group

The following `update-group` example changes the name of the IAM group `Test` to `Test-1`.

```
aws iam update-group \
    --group-name Test \
    --new-group-name Test-1
```

This command produces no output.

For more information, see Renaming an IAM user group in the *AWS IAM User Guide*.

- For API details, see UpdateGroup in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example renames the IAM group `Testers` to `AppTesters`.

```
Update-IAMGroup -GroupName Testers -NewGroupName AppTesters
```

Example 2: This example changes the path of the IAM group `AppTesters` to `/Org1/Org2/`. This changes the ARN for the group to `arn:aws:iam::123456789012:group/Org1/Org2/AppTesters`.

```
Update-IAMGroup -GroupName AppTesters -NewPath /Org1/Org2/
```

- For API details, see UpdateGroup in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example renames the IAM group `Testers` to `AppTesters`.

```
Update-IAMGroup -GroupName Testers -NewGroupName AppTesters
```

Example 2: This example changes the path of the IAM group `AppTesters` to `/Org1/Org2/`. This changes the ARN for the group to `arn:aws:iam::123456789012:group/Org1/Org2/AppTesters`.

```
Update-IAMGroup -GroupName AppTesters -NewPath /Org1/Org2/
```

- For API details, see UpdateGroup in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateLoginProfile` with a CLI

The following code examples show how to use `UpdateLoginProfile`.

CLI

AWS CLI

To update the password for an IAM user

The following `update-login-profile` example creates a new password for the IAM user named Bob.

```
aws iam update-login-profile \
    --user-name Bob \
    --password <password>
```

This command produces no output.

To set a password policy for the account, use the `update-account-password-policy` command. If the new password violates the account password policy, the command returns a `PasswordPolicyViolation` error.

If the account password policy allows them to, IAM users can change their own passwords using the `change-password` command.

Store the password in a secure place. If the password is lost, it cannot be recovered, and you must create a new one using the `create-login-profile` command.

For more information, see Managing passwords for IAM users in the *AWS IAM User Guide*.

- For API details, see UpdateLoginProfile in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example sets a new temporary password for the IAM user Bob, and requires the user to change the password the next time the user signs in.

```
Update-IAMLoginProfile -UserName Bob -Password "P@ssw0rd1234" -
PasswordResetRequired $true
```

- For API details, see UpdateLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example sets a new temporary password for the IAM user Bob, and requires the user to change the password the next time the user signs in.

```
Update-IAMLoginProfile -UserName Bob -Password "P@ssw0rd1234" -
PasswordResetRequired $true
```

- For API details, see UpdateLoginProfile in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateOpenIdConnectProviderThumbprint` with a CLI

The following code examples show how to use `UpdateOpenIdConnectProviderThumbprint`.

CLI

AWS CLI

To replace the existing list of server certificate thumbprints with a new list

The following `update-open-id-connect-provider-thumbprint` example updates the certificate thumbprint list for the OIDC provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com` to use a new thumbprint.

```
aws iam update-open-id-connect-provider-thumbprint \
    --open-id-connect-provider-arn arn:aws:iam::123456789012:oidc-provider/
example.oidcprovider.com \
    --thumbprint-list 7359755EXAMPLEabc3060bce3EXAMPLEec4542a3
```

If your OIDC provider's discovery and JWKS endpoints use different certificates, include thumbprints for both endpoints in the `--thumbprint-list` value.

This command produces no output.

For more information, see Creating OpenID Connect (OIDC) identity providers in the *AWS IAM User Guide*.

- For API details, see UpdateOpenIdConnectProviderThumbprint in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example updates the certificate thumbprint list for the OIDC provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com` to use a new thumbprint. The OIDC provider shares the new value when the certificate that is associated with the provider changes.

```
Update-IAMOpenIDConnectProviderThumbprint -OpenIDConnectProviderArn
 arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com -ThumbprintList
 7359755EXAMPLEabc3060bce3EXAMPLEec4542a3
```

- For API details, see UpdateOpenIdConnectProviderThumbprint in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example updates the certificate thumbprint list for the OIDC provider whose ARN is `arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com` to use a new thumbprint. The OIDC provider shares the new value when the certificate that is associated with the provider changes.

```
Update-IAMOpenIDConnectProviderThumbprint -OpenIDConnectProviderArn
 arn:aws:iam::123456789012:oidc-provider/example.oidcprovider.com -ThumbprintList
 7359755EXAMPLEabc3060bce3EXAMPLEec4542a3
```

- For API details, see UpdateOpenIdConnectProviderThumbprint in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateRole` with a CLI

The following code examples show how to use `UpdateRole`.

CLI

AWS CLI

To change an IAM role's description or session duration

The following `update-role` example changes the description of the IAM role `production-role` to `Main production role` and sets the maximum session duration to 12 hours.

```
aws iam update-role \
    --role-name production-role \
    --description 'Main production role' \
    --max-session-duration 43200
```

This command produces no output.

For more information, see IAM role management in the *AWS IAM User Guide*.

- For API details, see UpdateRole in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example updates the role description and the maximum session duration value(in seconds) for which a role's session can be requested.

```
Update-IAMRole -RoleName MyRoleName -Description "My testing role" -
MaxSessionDuration 43200
```

- For API details, see UpdateRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example updates the role description and the maximum session duration value(in seconds) for which a role's session can be requested.

```
Update-IAMRole -RoleName MyRoleName -Description "My testing role" -
MaxSessionDuration 43200
```

- For API details, see UpdateRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateRoleDescription` with a CLI

The following code examples show how to use `UpdateRoleDescription`.

CLI

AWS CLI

To change an IAM role's description

The following `update-role-description` example changes the description of the IAM role `production-role` to `Main production role`.

```
aws iam update-role-description \
    --role-name production-role \
    --description 'Main production role'
```

Output:

```
{
    "Role": {
        "Path": "/",
        "RoleName": "production-role",
        "RoleId": "AROA1234567890EXAMPLE",
        "Arn": "arn:aws:iam::123456789012:role/production-role",
        "CreateDate": "2017-12-06T17:16:37+00:00",
        "AssumeRolePolicyDocument": {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::123456789012:root"
                    },
                    "Action": "sts:AssumeRole",
                    "Condition": {}
                }
            ]
        },
```

```
        "Description": "Main production role"
    }
}
```

For more information, see IAM role management in the *AWS IAM User Guide*.

- For API details, see UpdateRoleDescription in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example updates the description of an IAM role in your account.

```
Update-IAMRoleDescription -RoleName MyRoleName -Description "My testing role"
```

- For API details, see UpdateRoleDescription in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example updates the description of an IAM role in your account.

```
Update-IAMRoleDescription -RoleName MyRoleName -Description "My testing role"
```

- For API details, see UpdateRoleDescription in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateSamlProvider` with a CLI

The following code examples show how to use `UpdateSamlProvider`.

CLI

AWS CLI

To update the metadata document for an existing SAML provider

The following `update-saml-provider` example updates the SAML provider in IAM whose ARN is `arn:aws:iam::123456789012:saml-provider/SAMLADFS` with a new SAML metadata document from the file `SAMLMetaData.xml`.

```
aws iam update-saml-provider \
    --saml-metadata-document file://SAMLMetaData.xml \
    --saml-provider-arn arn:aws:iam::123456789012:saml-provider/SAMLADFS
```

Contents of `SAMLMetaData.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
 entityID="https://adfs.example.com/adfs/services/trust">
    <md:IDPSSODescriptor
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <md:KeyDescriptor use="signing">
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:X509Data>
                    <ds:X509Certificate>MIIC8DCCAdigAwIBAgIQEXAMPLE...</
ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </md:KeyDescriptor>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</
md:NameIDFormat>
        <md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</
md:NameIDFormat>
        <md:SingleSignOnService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect" Location="https://
adfs.example.com/adfs/ls/"/>
        <md:SingleSignOnService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://
adfs.example.com/adfs/ls/"/>
    </md:IDPSSODescriptor>
</md:EntityDescriptor>
```

Output:

```
{
    "SAMLProviderArn": "arn:aws:iam::123456789012:saml-provider/SAMLADFS"
}
```

For more information, see Creating IAM SAML identity providers in the *AWS IAM User Guide*.

- For API details, see UpdateSamlProvider in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example updates the SAML provider in IAM whose ARN is `arn:aws:iam::123456789012:saml-provider/SAMLADFS` with a new SAML metadata document from the file `SAMLMetaData.xml`. Note that you must use the `-Raw` switch parameter to successfully process the contents of the JSON file.

```
Update-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/
SAMLADFS -SAMLMetadataDocument (Get-Content -Raw SAMLMetaData.xml)
```

- For API details, see UpdateSamlProvider in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example updates the SAML provider in IAM whose ARN is `arn:aws:iam::123456789012:saml-provider/SAMLADFS` with a new SAML metadata document from the file `SAMLMetaData.xml`. Note that you must use the `-Raw` switch parameter to successfully process the contents of the JSON file.

```
Update-IAMSAMLProvider -SAMLProviderArn arn:aws:iam::123456789012:saml-provider/
SAMLADFS -SAMLMetadataDocument (Get-Content -Raw SAMLMetaData.xml)
```

- For API details, see UpdateSamlProvider in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateServerCertificate` with an AWS SDK or CLI

The following code examples show how to use `UpdateServerCertificate`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::updateServerCertificate(const Aws::String
 &currentCertificateName,
                                           const Aws::String &newCertificateName,
                                           const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);
    Aws::IAM::Model::UpdateServerCertificateRequest request;
    request.SetServerCertificateName(currentCertificateName);
    request.SetNewServerCertificateName(newCertificateName);

    auto outcome = iam.UpdateServerCertificate(request);
    bool result = true;
    if (outcome.IsSuccess()) {
        std::cout << "Server certificate " << currentCertificateName
                  << " successfully renamed as " << newCertificateName
                  << std::endl;
    }
    else {
        if (outcome.GetError().GetErrorType() !=
 Aws::IAM::IAMErrors::NO_SUCH_ENTITY) {
            std::cerr << "Error changing name of server certificate " <<
                      currentCertificateName << " to " << newCertificateName <<
 ":" <<
                      outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
        else {
            std::cout << "Certificate '" << currentCertificateName
                      << "' not found." << std::endl;
        }
    }
```

```
    return result;
}
```

- For API details, see UpdateServerCertificate in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To change the path or name of a server certificate in your AWS account

The following `update-server-certificate` example changes the name of the certificate from `myServerCertificate` to `myUpdatedServerCertificate`. It also changes the path to `/cloudfront/` so that it can be accessed by the Amazon CloudFront service. You can see the results of the update by running the `list-server-certificates` command.

```
aws iam update-server-certificate \
    --server-certificate-name myServerCertificate \
    --new-server-certificate-name myUpdatedServerCertificate \
    --new-path /cloudfront/
```

This command produces no output.

For more information, see Managing server certificates in IAM in the *AWS IAM User Guide*.

- For API details, see UpdateServerCertificate in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Update a server certificate.

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
  const command = new UpdateServerCertificateCommand({
    ServerCertificateName: currentName,
    NewServerCertificateName: newName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see UpdateServerCertificate in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};
```

```
iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see UpdateServerCertificate in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example renames the certificate named `MyServerCertificate` to `MyRenamedServerCertificate`.

```
Update-IAMServerCertificate -ServerCertificateName MyServerCertificate -
NewServerCertificateName MyRenamedServerCertificate
```

Example 2: This example moves the certificate named `MyServerCertificate` to the path `/Org1/Org2/`. This changes the ARN for the resource to `arn:aws:iam::123456789012:server-certificate/Org1/Org2/MyServerCertificate`.

```
Update-IAMServerCertificate -ServerCertificateName MyServerCertificate -NewPath /
Org1/Org2/
```

- For API details, see UpdateServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example renames the certificate named `MyServerCertificate` to `MyRenamedServerCertificate`.

```
Update-IAMServerCertificate -ServerCertificateName MyServerCertificate -
NewServerCertificateName MyRenamedServerCertificate
```

Example 2: This example moves the certificate named `MyServerCertificate` to the path `/Org1/Org2/`. This changes the ARN for the resource to `arn:aws:iam::123456789012:server-certificate/Org1/Org2/MyServerCertificate`.

```
Update-IAMServerCertificate -ServerCertificateName MyServerCertificate -NewPath /
Org1/Org2/
```

- For API details, see UpdateServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List, update, and delete server certificates.

```
class ServerCertificateManager
  def initialize(iam_client, logger: Logger.new($stdout))
    @iam_client = iam_client
    @logger = logger
    @logger.progname = 'ServerCertificateManager'
  end

  # Creates a new server certificate.
  # @param name [String] the name of the server certificate
  # @param certificate_body [String] the contents of the certificate
  # @param private_key [String] the private key contents
  # @return [Boolean] returns true if the certificate was successfully created
  def create_server_certificate(name, certificate_body, private_key)
    @iam_client.upload_server_certificate({
                                            server_certificate_name: name,
                                            certificate_body: certificate_body,
                                            private_key: private_key
                                          })
```

```
    true
  rescue Aws::IAM::Errors::ServiceError => e
    puts "Failed to create server certificate: #{e.message}"
    false
  end

  # Lists available server certificate names.
  def list_server_certificate_names
    response = @iam_client.list_server_certificates

    if response.server_certificate_metadata_list.empty?
      @logger.info('No server certificates found.')
      return
    end

    response.server_certificate_metadata_list.each do |certificate_metadata|
      @logger.info("Certificate Name:
#{certificate_metadata.server_certificate_name}")
    end
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error listing server certificates: #{e.message}")
  end

  # Updates the name of a server certificate.
  def update_server_certificate_name(current_name, new_name)
    @iam_client.update_server_certificate(
      server_certificate_name: current_name,
      new_server_certificate_name: new_name
    )
    @logger.info("Server certificate name updated from '#{current_name}' to
'#{new_name}'.")
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error updating server certificate name: #{e.message}")
    false
  end

  # Deletes a server certificate.
  def delete_server_certificate(name)
    @iam_client.delete_server_certificate(server_certificate_name: name)
    @logger.info("Server certificate '#{name}' deleted.")
    true
  rescue Aws::IAM::Errors::ServiceError => e
    @logger.error("Error deleting server certificate: #{e.message}")
```

```
    false
  end
end
```

- For API details, see UpdateServerCertificate in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateSigningCertificate` with a CLI

The following code examples show how to use `UpdateSigningCertificate`.

CLI

AWS CLI

To activate or deactivate a signing certificate for an IAM user

The following `update-signing-certificate` example deactivates the specified signing certificate for the IAM user named Bob.

```
aws iam update-signing-certificate \
    --certificate-id TA7SMP42TDN5Z26OBPJE7EXAMPLE \
    --status Inactive \
    --user-name Bob
```

This command produces no output.

To get the ID for a signing certificate, use the `list-signing-certificates` command.

For more information, see Manage signing certificates in the *Amazon EC2 User Guide*.

- For API details, see UpdateSigningCertificate in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example updates the certificate that is associated with the IAM user named Bob and whose certificate ID si `Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU` to mark it as inactive.

```
Update-IAMSigningCertificate -CertificateId Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU -
UserName Bob -Status Inactive
```

- For API details, see UpdateSigningCertificate in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example updates the certificate that is associated with the IAM user named Bob and whose certificate ID si `Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU` to mark it as inactive.

```
Update-IAMSigningCertificate -CertificateId Y3EK7RMEXAMPLESV33FCREXAMPLEMJLU -
UserName Bob -Status Inactive
```

- For API details, see UpdateSigningCertificate in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateUser` with an AWS SDK or CLI

The following code examples show how to use `UpdateUser`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create read-only and read-write users

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::IAM::updateUser(const Aws::String &currentUserName,
                             const Aws::String &newUserName,
                             const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::IAM::IAMClient iam(clientConfig);

    Aws::IAM::Model::UpdateUserRequest request;
    request.SetUserName(currentUserName);
    request.SetNewUserName(newUserName);

    auto outcome = iam.UpdateUser(request);
    if (outcome.IsSuccess()) {
        std::cout << "IAM user " << currentUserName <<
                  " successfully updated with new user name " << newUserName <<
                  std::endl;
    }
    else {
        std::cerr << "Error updating user name for IAM user " << currentUserName
 <<
                  ":" << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see UpdateUser in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To change an IAM user's name

The following `update-user` example changes the name of the IAM user Bob to `Robert`.

```
aws iam update-user \
    --user-name Bob \
    --new-user-name Robert
```

This command produces no output.

For more information, see Renaming an IAM user group in the *AWS IAM User Guide*.

- For API details, see UpdateUser in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.IamException;
import software.amazon.awssdk.services.iam.model.UpdateUserRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
```

```
public class UpdateUser {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <curName> <newName>\s

                Where:
                    curName - The current user name.\s
                    newName - An updated user name.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String curName = args[0];
        String newName = args[1];
        Region region = Region.AWS_GLOBAL;
        IamClient iam = IamClient.builder()
                .region(region)
                .build();

        updateIAMUser(iam, curName, newName);
        System.out.println("Done");
        iam.close();
    }

    public static void updateIAMUser(IamClient iam, String curName, String
 newName) {
        try {
            UpdateUserRequest request = UpdateUserRequest.builder()
                    .userName(curName)
                    .newUserName(newName)
                    .build();

            iam.updateUser(request);
            System.out.printf("Successfully updated user to username %s",
 newName);

        } catch (IamException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }
}
```

- For API details, see UpdateUser in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Update the user.

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
  const command = new UpdateUserCommand({
    UserName: currentUserName,
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see UpdateUser in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see UpdateUser in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun updateIAMUser(
    curName: String?,
    newName: String?,
) {
    val request =
        UpdateUserRequest {
            userName = curName
            newUserName = newName
        }

    IamClient.fromEnvironment { region = "AWS_GLOBAL" }.use { iamClient ->
        iamClient.updateUser(request)
        println("Successfully updated user to $newName")
    }
}
```

- For API details, see UpdateUser in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example renames the IAM user Bob to Robert.

```
Update-IAMUser -UserName Bob -NewUserName Robert
```

Example 2: This example changes the path of the IAM User Bob to /Org1/Org2/, which effectively changes the ARN for the user to arn:aws:iam::123456789012:user/Org1/Org2/bob.

```
Update-IAMUser -UserName Bob -NewPath /Org1/Org2/
```

- For API details, see UpdateUser in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example renames the IAM user Bob to Robert.

```
Update-IAMUser -UserName Bob -NewUserName Robert
```

Example 2: This example changes the path of the IAM User Bob to /Org1/Org2/, which effectively changes the ARN for the user to arn:aws:iam::123456789012:user/Org1/Org2/bob.

```
Update-IAMUser -UserName Bob -NewPath /Org1/Org2/
```

- For API details, see UpdateUser in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def update_user(user_name, new_user_name):
    """
    Updates a user's name.

    :param user_name: The current name of the user to update.
    :param new_user_name: The new name to assign to the user.
    :return: The updated user.
    """
    try:
        user = iam.User(user_name)
        user.update(NewUserName=new_user_name)
```

```
            logger.info("Renamed %s to %s.", user_name, new_user_name)
        except ClientError:
            logger.exception("Couldn't update name for user %s.", user_name)
            raise
        return user
```

- For API details, see UpdateUser in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
 # Updates an IAM user's name
 #
 # @param current_name [String] The current name of the user
 # @param new_name [String] The new name of the user
 def update_user_name(current_name, new_name)
   @iam_client.update_user(user_name: current_name, new_user_name: new_name)
   true
 rescue StandardError => e
   @logger.error("Error updating user name from '#{current_name}' to
'#{new_name}': #{e.message}")
   false
 end
```

- For API details, see UpdateUser in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_iam->updateuser(
      iv_username = iv_user_name
      iv_newusername = iv_new_user_name ).
    MESSAGE 'User updated successfully.' TYPE 'I'.
  CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'User does not exist.' TYPE 'E'.
  CATCH /aws1/cx_iamentityalrdyexex.
    MESSAGE 'New user name already exists.' TYPE 'E'.
ENDTRY.
```

- For API details, see UpdateUser in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UploadServerCertificate` with an AWS SDK or CLI

The following code examples show how to use `UploadServerCertificate`.

CLI

AWS CLI

To upload a server certificate to your AWS account

The following `upload-server-certificate` example uploads a server certificate to your AWS account. In this example, the certificate is in the file `public_key_cert_file.pem`, the associated private key is in the file `my_private_key.pem`, and the certificate chain

provided by the certificate authority (CA) is in the `my_certificate_chain_file.pem` file. When the file has finished uploading, it is available under the name *myServerCertificate*. Parameters that begin with `file://` tell the command to read the contents of the file and use that as the parameter value instead of the file name itself.

```
aws iam upload-server-certificate \
    --server-certificate-name myServerCertificate \
    --certificate-body file://public_key_cert_file.pem \
    --private-key file://my_private_key.pem \
    --certificate-chain file://my_certificate_chain_file.pem
```

Contents of `public_key_cert_file.pem`:

```
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAEXAMPLE...
-----END CERTIFICATE-----
```

Contents of `my_private_key.pem`:

```
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAAEXAMPLE...
-----END PRIVATE KEY-----
```

Contents of `my_certificate_chain_file.pem`:

```
-----BEGIN CERTIFICATE-----
MIIEkjCCA3qgAwIBAgIQEXAMPLE...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDSjCCAjKgAwIBAgIQEXAMPLE...
-----END CERTIFICATE-----
```

Output:

```
{
    "ServerCertificateMetadata": {
        "Path": "/",
        "ServerCertificateName": "myServerCertificate",
        "ServerCertificateId": "ASCAEXAMPLE123EXAMPLE",
        "Arn": "arn:aws:iam::1234567989012:server-certificate/
myServerCertificate",
```

```
        "UploadDate": "2019-04-22T21:13:44+00:00",
        "Expiration": "2019-10-15T22:23:16+00:00"
    }
}
```

For more information, see Manage server certificates in IAM in the *AWS IAM User Guide*.

- For API details, see UploadServerCertificate in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "node:path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command,
 or the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
```

```
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }

    throw err;
  }
};

/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};
```

- For API details, see UploadServerCertificate in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example uploads a new server certificate to the IAM account. The files containing the certificate body, the private key, and (optionally) the certificate chain must all be PEM encoded. Note that the parameters require the actual content of the files rather than the file names. You must use the `-Raw` switch parameter to successfully process the file contents.

```
Publish-IAMServerCertificate -ServerCertificateName MyTestCert -CertificateBody
 (Get-Content -Raw server.crt) -PrivateKey (Get-Content -Raw server.key)
```

Output:

```
Arn                 : arn:aws:iam::123456789012:server-certificate/MyTestCert
Expiration          : 1/14/2018 9:52:36 AM
Path                : /
ServerCertificateId : ASCAJIEXAMPLE7J7HQZYW
ServerCertificateName : MyTestCert
UploadDate          : 4/21/2015 11:14:16 AM
```

- For API details, see UploadServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example uploads a new server certificate to the IAM account. The files containing the certificate body, the private key, and (optionally) the certificate chain must all be PEM encoded. Note that the parameters require the actual content of the files rather than the file names. You must use the `-Raw` switch parameter to successfully process the file contents.

```
Publish-IAMServerCertificate -ServerCertificateName MyTestCert -CertificateBody
 (Get-Content -Raw server.crt) -PrivateKey (Get-Content -Raw server.key)
```

Output:

```
Arn                 : arn:aws:iam::123456789012:server-certificate/MyTestCert
Expiration          : 1/14/2018 9:52:36 AM
Path                : /
ServerCertificateId : ASCAJIEXAMPLE7J7HQZYW
ServerCertificateName : MyTestCert
UploadDate          : 4/21/2015 11:14:16 AM
```

- For API details, see UploadServerCertificate in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UploadSigningCertificate` with a CLI

The following code examples show how to use `UploadSigningCertificate`.

CLI

AWS CLI

To upload a signing certificate for an IAM user

The following `upload-signing-certificate` example uploads a signing certificate for the IAM user named Bob.

```
aws iam upload-signing-certificate \
    --user-name Bob \
    --certificate-body file://certificate.pem
```

Contents of `certificate.pem`:

```
-----BEGIN CERTIFICATE-----
MIICdzCCAeCgAwIBAgIGANc+EXAMPLE...
-----END CERTIFICATE-----
```

Output:

```
{
    "Certificate": {
        "UserName": "Bob",
        "Status": "Active",
        "CertificateBody": "-----BEGIN CERTIFICATE-----<certificate-body>-----END
 CERTIFICATE-----",
        "CertificateId": "TA7SMP42TDN5Z26OBPJE7EXAMPLE",
        "UploadDate": "2013-06-06T21:40:08.121Z"
    }
}
```

The certificate is in a file named *certificate.pem* in PEM format.

For more information, see Set up the Amazon EC2 AMI tools in the *Amazon EC2 User Guide*.

- For API details, see UploadSigningCertificate in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example uploads a new X.509 signing certificate and associates it with the IAM user named Bob. The file containing the certificate body is PEM encoded. The `CertificateBody` parameter requires the actual contents of the certificate file rather than the file name. You must use the `-Raw` switch parameter to successfully process the file.

```
Publish-IAMSigningCertificate -UserName Bob -CertificateBody (Get-Content -Raw
 SampleSigningCert.pem)
```

Output:

```
CertificateBody : -----BEGIN CERTIFICATE-----

 MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC

 VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6

 b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd

 BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN

 MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD

 VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z

 b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt
                  YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn
+a4GmWIWJ
                  21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/
f0wYK8m9T
                  rDHudUZg3qX4waLG5M43q7Wgc/
MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE

 Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
```

```
                  nUhVVxYUntneD9+h8Mg9q6q
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb

 FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
                  NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
                  -----END CERTIFICATE-----
CertificateId   : Y3EK7RMEXAMPLESV33FCEXAMPLEHMJLU
Status          : Active
UploadDate      : 4/20/2015 1:26:01 PM
UserName        : Bob
```

- For API details, see [UploadSigningCertificate](#) in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example uploads a new X.509 signing certificate and associates it with the IAM user named Bob. The file containing the certificate body is PEM encoded. The `CertificateBody` parameter requires the actual contents of the certificate file rather than the file name. You must use the `-Raw` switch parameter to successfully process the file.

```
Publish-IAMSigningCertificate -UserName Bob -CertificateBody (Get-Content -Raw
 SampleSigningCert.pem)
```

Output:

```
CertificateBody : -----BEGIN CERTIFICATE-----

 MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC

 VVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6

 b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd

 BgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhcNMTEwNDI1MjA0NTIxWhcN

 MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYD

 VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25z

 b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFt
```

```
                    YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn
+a4GmWIWJ
                    21uUSfwfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/
f0wYK8m9T
                    rDHudUZg3qX4waLG5M43q7Wgc/
MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE

 Ibb3OhjZnzcvQAaRHhdlQWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
                    nUhVVxYUntneD9+h8Mg9q6q
+auNKyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb

 FFBjvSfpJIlJ00zbhNYS5f6GuoEDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTb
                    NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
                    -----END CERTIFICATE-----
CertificateId   : Y3EK7RMEXAMPLESV33FCEXAMPLEHMJLU
Status          : Active
UploadDate      : 4/20/2015 1:26:01 PM
UserName        : Bob
```

- For API details, see UploadSigningCertificate in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for IAM using AWS SDKs

The following code examples show you how to implement common scenarios in IAM with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within IAM or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Build and manage a resilient service using an AWS SDK
- Configure container service connectivity
- Create a container task for the serverless launch type

- Create a rest API with function proxy integration
- Create read-only and read-write IAM users using an AWS SDK
- Creating a container service for virtual machine instances
- Creating a managed monitoring workspace
- Creating a monitoring dashboard with function name as a variable
- Creating your first serverless function
- Get started with serverless data warehouses using the CLI
- Getting started with configuration management
- Getting started with internet of things device protection
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with provisioned data warehouse clusters
- Getting started with workflow orchestration
- Manage IAM access keys using an AWS SDK
- Manage IAM policies using an AWS SDK
- Manage IAM roles using an AWS SDK
- Manage your IAM account using an AWS SDK
- Moving hardcoded secrets to secure secret storage
- Allows the AWS Compute Optimizer Automation feature to apply recommended actions
- Policy to enable Automation across your organization
- Policy to enable Automation for your account
- Policy to grant full access to Compute Optimizer Automation for a management account of an organization
- Policy to grant full access to Compute Optimizer Automation for standalone AWS accounts
- Policy to grant read-only access to Compute Optimizer Automation for a management account of an organization
- Policy to grant read-only access to Compute Optimizer Automation for standalone AWS accounts
- Policy to grants service-linked role permissions for Compute Optimization Automation

- Roll back an IAM policy version using an AWS SDK
- Run CPU stress tests on virtual machine instances using fault injection
- Set up Attribute-Based Access Control for DynamoDB using AWS Command Line Interface v2
- Setting up systems management
- Using property variables in monitoring dashboards to monitor multiple serverless functions
- Work with DynamoDB Streams and Time-to-Live using AWS Command Line Interface v2
- Work with the IAM Policy Builder API using an AWS SDK

Build and manage a resilient service using an AWS SDK

The following code examples show how to create a load-balanced web service that returns book, movie, and song recommendations. The example shows how the service responds to failures, and how to restructure the service for more resilience when failures occur.

- Use an Amazon EC2 Auto Scaling group to create Amazon Elastic Compute Cloud (Amazon EC2) instances based on a launch template and to keep the number of instances in a specified range.
- Handle and distribute HTTP requests with Elastic Load Balancing.
- Monitor the health of instances in an Auto Scaling group and forward requests only to healthy instances.
- Run a Python web server on each EC2 instance to handle HTTP requests. The web server responds with recommendations and health checks.
- Simulate a recommendation service with an Amazon DynamoDB table.
- Control web server response to requests and health checks by updating AWS Systems Manager parameters.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
```

```
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
```

```
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper =
host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several
AWS resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways
to make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide
book, movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each
contain a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across
several Availability Zones.");
        Console.WriteLine(
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the
Auto Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
```

```
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs
'server_startup_script.sh' when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the
`server.py` script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to
'/' and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and
Systems Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2
instances, each in a different\n"
```

```
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3,
_autoScalerWrapper.GroupName, zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or
continue with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the
demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGro
protocol, port, defaultVpc.VpcId);

        await
_elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.Lo
subnetIds, targetGroup);
        await
_autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
```

```
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port,
 ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for
 your default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management
 Console.\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port,
 ipString);
                }
            }

            if (!sshPortIsOpen)
```

```
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
                }
            }
            loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
```

```
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
                          "to create situations where the web service fails, and
shows how using a resilient\n" +
                          "architecture can keep the web service running in spite
of these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
"this-is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service
fails, the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");
```

```
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in
the target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile
that contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
```

```
            "Now, sending a GET request to the load balancer endpoint returns
either a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo,
a deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not
for Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the
Auto Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load
balancer can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two
for the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by
an auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler
start a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);
```

```
        Console.WriteLine($"\nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a
successful recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the
load balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance
typically takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
"this-is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account,
we can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBal
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGr
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupNa
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you
might incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Create a class that wraps Auto Scaling and Amazon EC2 actions.

```
/// <summary>
```

```
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;
    private readonly ILogger<AutoScalerWrapper> _logger;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration,
        ILogger<AutoScalerWrapper> logger)
```

```
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;
        _logger = logger;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with
a specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to
the role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {
```

```
        var assumeRoleDoc = "{" +
                                    "\"Version\": \"2012-10-17\"," +
                                    "\"Statement\": [{" +
                                         "\"Effect\": \"Allow\"," +
                                         "\"Principal\": {" +
                                         "\"Service\": [" +
                                             "\"ec2.amazonaws.com\"" +
                                         "]" +
                                         "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                                    "}]" +
                                "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }
```

```
            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await
_amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
```

```
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
```

```
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto
Scaling.
    /// The launch template specifies a Bash script in its user data field that
runs after
    /// the instance is started. This script installs the Python packages and
starts a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to
create and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        try
        {
            await CreateKeyPair(_keyPairName);
            await CreateInstanceProfileWithName(_instancePolicyName,
_instanceRoleName,
```

```
                _instanceProfileName, instancePolicyPath);

            var startServerText = await File.ReadAllTextAsync(startupScriptPath);
            var plainTextBytes =
System.Text.Encoding.UTF8.GetBytes(startServerText);

            var amiLatest = await _amazonSsm.GetParameterAsync(
                new GetParameterRequest() { Name = _amiParam });
            var amiId = amiLatest.Parameter.Value;
            var launchTemplateResponse = await
_amazonEc2.CreateLaunchTemplateAsync(
                new CreateLaunchTemplateRequest()
                {
                    LaunchTemplateName = _launchTemplateName,
                    LaunchTemplateData = new RequestLaunchTemplateData()
                    {
                        InstanceType = _instanceType,
                        ImageId = amiId,
                        IamInstanceProfile =
                            new

LaunchTemplateIamInstanceProfileSpecificationRequest()
                            {
                                Name = _instanceProfileName
                            },
                        KeyName = _keyPairName,
                        UserData = System.Convert.ToBase64String(plainTextBytes)
                    }
                });
            return launchTemplateResponse.LaunchTemplate;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.AlreadyExistsException")
            {
                _logger.LogError($"Could not create the template, the name
{_launchTemplateName} already exists. " +
                                 $"Please try again with a unique name.");
            }

            throw;
        }
        catch (Exception ex)
```

```
        {
            _logger.LogError($"An error occurred while creating the template.:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2
 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        try
        {
            var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
                new DescribeAvailabilityZonesRequest());
            return zoneResponse.AvailabilityZones.Select(z =>
 z.ZoneName).ToList();
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            _logger.LogError($"An Amazon EC2 error occurred while listing
 availability zones.: {ec2Exception.Message}");
            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while listing availability
 zones.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
```

```
   public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)
   {
       try
       {
           await _amazonAutoScaling.CreateAutoScalingGroupAsync(
               new CreateAutoScalingGroupRequest()
               {
                   AutoScalingGroupName = groupName,
                   AvailabilityZones = availabilityZones,
                   LaunchTemplate =
                       new
Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                       {
                           LaunchTemplateName = _launchTemplateName,
                           Version = "$Default"
                       },
                   MaxSize = groupSize,
                   MinSize = groupSize
               });
           Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with
size {groupSize}.");
       }
       catch (EntityAlreadyExistsException)
       {
           Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
       }
   }

   /// <summary>
   /// Get the default VPC for the account.
   /// </summary>
   /// <returns>The default VPC object.</returns>
   public async Task<Vpc> GetDefaultVpc()
   {
       try
       {
           var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
               new DescribeVpcsRequest()
               {
                   Filters = new List<Amazon.EC2.Model.Filter>()
                   {
                       new("is-default", new List<string>() { "true" })
```

```
                    }
                });
            return vpcResponse.Vpcs[0];
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "UnauthorizedOperation")
            {
                _logger.LogError(ec2Exception, $"You do not have the necessary
permissions to describe VPCs.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while describing the vpcs.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        try
        {
            var subnets = new List<Subnet>();
            var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
                new DescribeSubnetsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("vpc-id", new List<string>() { vpcId }),
                        new("availability-zone", availabilityZones),
                        new("default-for-az", new List<string>() { "true" })
                    }
                });
```

```
            // Get the entire list using the paginator.
            await foreach (var subnet in subnetPaginator.Subnets)
            {
                subnets.Add(subnet);
            }

            return subnets;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidVpcID.NotFound")
            {
                _logger.LogError(ec2Exception, $"The specified VPC ID {vpcId}
does not exist.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while describing the
subnets.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonEC2Exception ec2Exception)
        {
```

```
            if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.NotFoundException")
            {
                _logger.LogError(
                    $"Could not delete the template, the name
{_launchTemplateName} was not found.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while deleting the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the
role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await
_amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
```

```
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }
```

```
   /// <summary>
   /// Get the instance profile association data for an instance.
   /// </summary>
   /// <param name="instanceId">The Id of the instance.</param>
   /// <returns>Instance profile associations data.</returns>
   public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
   {
       try
       {
           var response = await
_amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
               new DescribeIamInstanceProfileAssociationsRequest()
               {
                   Filters = new List<Amazon.EC2.Model.Filter>()
                   {
                       new("instance-id", new List<string>() { instanceId })
                   },
               });
           return response.IamInstanceProfileAssociations[0];
       }
       catch (AmazonEC2Exception ec2Exception)
       {
           if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
           {
               _logger.LogError(ec2Exception, $"Instance {instanceId} not
found");
           }

           throw;
       }
       catch (Exception ex)
       {
           _logger.LogError(ex, $"An error occurred while creating the
template.: {ex.Message}");
           throw;
       }
   }

   /// <summary>
   /// Replace the profile associated with a running instance. After the profile
is replaced, the instance
```

```
   /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
   /// used to restart the Python web server.
   /// </summary>
   /// <param name="instanceId">The Id of the instance to update.</param>
   /// <param name="credsProfileName">The name of the new profile to associate
with the specified instance.</param>
   /// <param name="associationId">The Id of the existing profile association
for the instance.</param>
   /// <returns>Async task.</returns>
   public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
   {
       try
       {
           await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
               new ReplaceIamInstanceProfileAssociationRequest()
               {
                   AssociationId = associationId,
                   IamInstanceProfile = new IamInstanceProfileSpecification()
                   {
                       Name = credsProfileName
                   }
               });
           // Allow time before resetting.
           Thread.Sleep(25000);

           await _amazonEc2.RebootInstancesAsync(
               new RebootInstancesRequest(new List<string>() { instanceId }));
           Thread.Sleep(25000);
           var instanceReady = false;
           var retries = 5;
           while (retries-- > 0 && !instanceReady)
           {
               var instancesPaginator =
                   _amazonSsm.Paginators.DescribeInstanceInformation(
                       new DescribeInstanceInformationRequest());
               // Get the entire list using the paginator.
               await foreach (var instance in
instancesPaginator.InstanceInformationList)
               {
                   instanceReady = instance.InstanceId == instanceId;
                   if (instanceReady)
                   {
```

```
                        break;
                    }
                }
            }
            Console.WriteLine("Waiting for instance to be running.");
            await WaitForInstanceState(instanceId, InstanceStateName.Running);
            Console.WriteLine("Instance ready.");
            Console.WriteLine($"Sending restart command to instance
{instanceId}");
            await _amazonSsm.SendCommandAsync(
                new SendCommandRequest()
                {
                    InstanceIds = new List<string>() { instanceId },
                    DocumentName = "AWS-RunShellScript",
                    Parameters = new Dictionary<string, List<string>>()
                    {
                        {
                            "commands",
                            new List<string>() { "cd / && sudo python3 server.py
80" }
                        }
                    }
                });
            Console.WriteLine($"Restarted the web server on instance
{instanceId}");
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
            {
                _logger.LogError(ec2Exception, $"Instance {instanceId} not
found");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while replacing the
template.: {ex.Message}");
            throw;
        }
    }
```

```
    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await
_amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for
{instanceId}. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
```

```
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string
groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
```

```
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the
calling computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP
address while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
```

```
   /// <param name="ipAddress">This computer's IP address.</param>
   /// <returns>True if the ip address is allowed on the group.</returns>
   public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
   {
       var portIsOpen = false;
       foreach (var ipPermission in group.IpPermissions)
       {
           if (ipPermission.FromPort == port)
           {
               foreach (var ipRange in ipPermission.Ipv4Ranges)
               {
                   var cidr = ipRange.CidrIp;
                   if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                   {
                       portIsOpen = true;
                   }
               }

               if (ipPermission.PrefixListIds.Any())
               {
                   portIsOpen = true;
               }

               if (!portIsOpen)
               {
                   Console.WriteLine("The inbound rule does not appear to be
open to either this computer's IP\n" +
                                     "address, to all IP addresses (0.0.0.0/0),
or to a prefix list ID.");
               }
               else
               {
                   break;
               }
           }
       }

       return portIsOpen;
   }

   /// <summary>
   /// Add an ingress rule to the specified security group that allows access on
the
```

```
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName,
 string targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
```

```
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>
    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
 InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is in the specified state.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEc2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
            Console.Write(". ");
        } while (!hasState);

        return hasState;
    }
}
```

Create a class that wraps Elastic Load Balancing actions.

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
```

```
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
 client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
 loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
```

```
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
```

```
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened
 times and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</
param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer
 exists.</param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }
```

```
   /// <summary>
   /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
   /// and forwards requests to the specified target group.
   /// </summary>
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await
_amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });

               var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

               loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
           }
           catch (LoadBalancerNotFoundException)
           {
```

```
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
```

```
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
```

```
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await
_amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn =
targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
```

```
}
```

Create a class that uses DynamoDB to simulate a recommendation service.

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books,
 movies, and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
```

```
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
                            AttributeName = "ItemId",
                            AttributeType = ScalarAttributeType.N
                        }
                    },
                KeySchema = new List<KeySchemaElement>()
                    {
                        new KeySchemaElement()
                        {
                            AttributeName = "MediaType",
                            KeyType = KeyType.HASH
                        },
                        new KeySchemaElement()
                        {
                            AttributeName = "ItemId",
                            KeyType = KeyType.RANGE
                        }
                    },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5
                }
            };
            await _amazonDynamoDb.CreateTableAsync(createRequest);

            // Wait until the table is ACTIVE and then report success.
            Console.Write("\nWaiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = tableName
            };

            TableStatus status;
            do
```

```
            {
                Thread.Sleep(2000);

                var describeTableResponse = await
 _amazonDynamoDb.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await
 File.ReadAllTextAsync(recommendationsPath);
        var records =

 JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }
```

```
    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Create a class that wraps Systems Manager actions.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement
 _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-
architecture-table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";
```

```
    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 true });
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
public class Main {

    public static final String fileName = "C:\\AWS\\resworkflow\
\recommendations.json"; // Modify file location.
    public static final String tableName = "doc-example-recommendation-service";
    public static final String startScript = "C:\\AWS\\resworkflow\
\server_startup_script.sh"; // Modify file location.
    public static final String policyFile = "C:\\AWS\\resworkflow\
\instance_policy.json"; // Modify file location.
    public static final String ssmJSON = "C:\\AWS\\resworkflow\
\ssm_only_policy.json"; // Modify file location.
    public static final String failureResponse = "doc-example-resilient-
architecture-failure-response";
    public static final String healthCheck = "doc-example-resilient-architecture-
health-check";
    public static final String templateName = "doc-example-resilience-template";
    public static final String roleName = "doc-example-resilience-role";
    public static final String policyName = "doc-example-resilience-pol";
    public static final String profileName = "doc-example-resilience-prof";

    public static final String badCredsProfileName = "doc-example-resilience-
prof-bc";

    public static final String targetGroupName = "doc-example-resilience-tg";
    public static final String autoScalingGroupName = "doc-example-resilience-
group";
    public static final String lbName = "doc-example-resilience-lb";
    public static final String protocol = "HTTP";
    public static final int port = 80;
```

```
   public static final String DASHES = new String(new char[80]).replace("\0",
"-");

   public static void main(String[] args) throws IOException,
InterruptedException {
       Scanner in = new Scanner(System.in);
       Database database = new Database();
       AutoScaler autoScaler = new AutoScaler();
       LoadBalancer loadBalancer = new LoadBalancer();

       System.out.println(DASHES);
       System.out.println("Welcome to the demonstration of How to Build and
Manage a Resilient Service!");
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("A - SETUP THE RESOURCES");
       System.out.println("Press Enter when you're ready to start deploying
resources.");
       in.nextLine();
       deploy(loadBalancer);
       System.out.println(DASHES);
       System.out.println(DASHES);
       System.out.println("B - DEMO THE RESILIENCE FUNCTIONALITY");
       System.out.println("Press Enter when you're ready.");
       in.nextLine();
       demo(loadBalancer);
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("C - DELETE THE RESOURCES");
       System.out.println("""
               This concludes the demo of how to build and manage a resilient
service.
               To keep things tidy and to avoid unwanted charges on your
account, we can clean up all AWS resources
               that were created for this demo.
               """);

       System.out.println("\n Do you want to delete the resources (y/n)? ");
       String userInput = in.nextLine().trim().toLowerCase(); // Capture user
input

       if (userInput.equals("y")) {
```

```
            // Delete resources here
            deleteResources(loadBalancer, autoScaler, database);
            System.out.println("Resources deleted.");
        } else {
            System.out.println("""
                    Okay, we'll leave the resources intact.
                    Don't forget to delete them when you're done with them or you
might incur unexpected charges.
                    """);
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The example has completed. ");
        System.out.println("\n Thanks for watching!");
        System.out.println(DASHES);
    }

    // Deletes the AWS resources used in this example.
    private static void deleteResources(LoadBalancer loadBalancer, AutoScaler
autoScaler, Database database)
            throws IOException, InterruptedException {
        loadBalancer.deleteLoadBalancer(lbName);
        System.out.println("*** Wait 30 secs for resource to be deleted");
        TimeUnit.SECONDS.sleep(30);
        loadBalancer.deleteTargetGroup(targetGroupName);
        autoScaler.deleteAutoScaleGroup(autoScalingGroupName);
        autoScaler.deleteRolesPolicies(policyName, roleName, profileName);
        autoScaler.deleteTemplate(templateName);
        database.deleteTable(tableName);
    }

    private static void deploy(LoadBalancer loadBalancer) throws
InterruptedException, IOException {
        Scanner in = new Scanner(System.in);
        System.out.println(
                """
                        For this demo, we'll use the AWS SDK for Java (v2) to
create several AWS resources
                        to set up a load-balanced web service endpoint and
explore some ways to make it resilient
                        against various kinds of failures.

                        Some of the resources create by this demo are:
```

```
                        \t* A DynamoDB table that the web service depends on to
provide book, movie, and song recommendations.
                        \t* An EC2 launch template that defines EC2 instances
that each contain a Python web server.
                        \t* An EC2 Auto Scaling group that manages EC2 instances
across several Availability Zones.
                        \t* An Elastic Load Balancing (ELB) load balancer that
targets the Auto Scaling group to distribute requests.
                        """);

        System.out.println("Press Enter when you're ready.");
        in.nextLine();
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("Creating and populating a DynamoDB table named " +
tableName);
        Database database = new Database();
        database.createTable(tableName, fileName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("""
                Creating an EC2 launch template that runs '{startup_script}' when
an instance starts.
                This script starts a Python web server defined in the `server.py`
script. The web server
                listens to HTTP requests on port 80 and responds to requests to
'/' and to '/healthcheck'.
                For demo purposes, this server is run as the root user. In
production, the best practice is to
                run a web server, such as Apache, with least-privileged
credentials.

                The template also defines an IAM policy that each instance uses
to assume a role that grants
                permissions to access the DynamoDB recommendation table and
Systems Manager parameters
                that control the flow of the demo.
                """);

        LaunchTemplateCreator templateCreator = new LaunchTemplateCreator();
        templateCreator.createTemplate(policyFile, policyName, profileName,
startScript, templateName, roleName);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
                "Creating an EC2 Auto Scaling group that maintains three EC2
instances, each in a different Availability Zone.");
        System.out.println("*** Wait 30 secs for the VPC to be created");
        TimeUnit.SECONDS.sleep(30);
        AutoScaler autoScaler = new AutoScaler();
        String[] zones = autoScaler.createGroup(3, templateName,
autoScalingGroupName);

        System.out.println("""
                At this point, you have EC2 instances created. Once each instance
starts, it listens for
                HTTP requests. You can see these instances in the console or
continue with the demo.
                Press Enter when you're ready to continue.
                """);

        in.nextLine();
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("Creating variables that control the flow of the
demo.");
        ParameterHelper paramHelper = new ParameterHelper();
        paramHelper.reset();
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("""
                Creating an Elastic Load Balancing target group and load
balancer. The target group
                defines how the load balancer connects to instances. The load
balancer provides a
                single endpoint where clients connect and dispatches requests to
instances in the group.
                """);

        String vpcId = autoScaler.getDefaultVPC();
        List<Subnet> subnets = autoScaler.getSubnets(vpcId, zones);
        System.out.println("You have retrieved a list with " + subnets.size() + "
subnets");
```

```
        String targetGroupArn = loadBalancer.createTargetGroup(protocol, port,
vpcId, targetGroupName);
        String elbDnsName = loadBalancer.createLoadBalancer(subnets,
targetGroupArn, lbName, port, protocol);
        autoScaler.attachLoadBalancerTargetGroup(autoScalingGroupName,
targetGroupArn);
        System.out.println("Verifying access to the load balancer endpoint...");
        boolean wasSuccessul =
loadBalancer.verifyLoadBalancerEndpoint(elbDnsName);
        if (!wasSuccessul) {
            System.out.println("Couldn't connect to the load balancer, verifying
that the port is open...");
            CloseableHttpClient httpClient = HttpClients.createDefault();

            // Create an HTTP GET request to "http://checkip.amazonaws.com"
            HttpGet httpGet = new HttpGet("http://checkip.amazonaws.com");
            try {
                // Execute the request and get the response
                HttpResponse response = httpClient.execute(httpGet);

                // Read the response content.
                String ipAddress =
IOUtils.toString(response.getEntity().getContent(),
StandardCharsets.UTF_8).trim();

                // Print the public IP address.
                System.out.println("Public IP Address: " + ipAddress);
                GroupInfo groupInfo = autoScaler.verifyInboundPort(vpcId, port,
ipAddress);
                if (!groupInfo.isPortOpen()) {
                    System.out.println("""
                            For this example to work, the default security group
for your default VPC must
                            allow access from this computer. You can either add
it automatically from this
                            example or add it yourself using the AWS Management
Console.
                            """);

                    System.out.println(
                            "Do you want to add a rule to security group " +
groupInfo.getGroupName() + " to allow");
                    System.out.println("inbound traffic on port " + port + " from
your computer's IP address (y/n) ");
```

```
                    String ans = in.nextLine();
                    if ("y".equalsIgnoreCase(ans)) {
                        autoScaler.openInboundPort(groupInfo.getGroupName(),
String.valueOf(port), ipAddress);
                        System.out.println("Security group rule added.");
                    } else {
                        System.out.println("No security group rule added.");
                    }
                }

            } catch (AutoScalingException e) {
                e.printStackTrace();
            }
        } else if (wasSuccessul) {
            System.out.println("Your load balancer is ready. You can access it by
browsing to:");
            System.out.println("\t http://" + elbDnsName);
        } else {
            System.out.println("Couldn't get a successful response from the load
balancer endpoint. Troubleshoot by");
            System.out.println("manually verifying that your VPC and security
group are configured correctly and that");
            System.out.println("you can successfully make a GET request to the
load balancer.");
        }

        System.out.println("Press Enter when you're ready to continue with the
demo.");
        in.nextLine();
    }

    // A method that controls the demo part of the Java program.
    public static void demo(LoadBalancer loadBalancer) throws IOException,
InterruptedException {
        ParameterHelper paramHelper = new ParameterHelper();
        System.out.println("Read the ssm_only_policy.json file");
        String ssmOnlyPolicy = readFileAsString(ssmJSON);

        System.out.println("Resetting parameters to starting values for demo.");
        paramHelper.reset();

        System.out.println(
                """
```

```
                        This part of the demonstration shows how to toggle
different parts of the system
                        to create situations where the web service fails, and
shows how using a resilient
                        architecture can keep the web service running in spite
of these failures.

                        At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.
                       """);
       demoChoices(loadBalancer);

       System.out.println(
               """
                        The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.
                        The table name is contained in a Systems Manager
parameter named self.param_helper.table.
                        To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.
                       """);
       paramHelper.put(paramHelper.tableName, "this-is-not-a-table");

       System.out.println(
               """
                        \nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as
                        healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.
                       """);
       demoChoices(loadBalancer);

       System.out.println(
               """
                       Instead of failing when the recommendation service fails,
the web service can return a static response.
                       While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.
                       """);
       paramHelper.put(paramHelper.failureResponse, "static");

       System.out.println("""
               Now, sending a GET request to the load balancer endpoint returns
a static response.
```

```
                The service still reports as healthy because health checks are
still shallow.
                """);
        demoChoices(loadBalancer);

        System.out.println("Let's reinstate the recommendation service.");
        paramHelper.put(paramHelper.tableName, paramHelper.dyntable);

        System.out.println("""
                Let's also substitute bad credentials for one of the instances in
the target group so that it can't
                access the DynamoDB recommendation table. We will get an instance
id value.
                """);

        LaunchTemplateCreator templateCreator = new LaunchTemplateCreator();
        AutoScaler autoScaler = new AutoScaler();

        // Create a new instance profile based on badCredsProfileName.
        templateCreator.createInstanceProfile(policyFile, policyName,
badCredsProfileName, roleName);
        String badInstanceId = autoScaler.getBadInstance(autoScalingGroupName);
        System.out.println("The bad instance id values used for this demo is " +
badInstanceId);

        String profileAssociationId =
autoScaler.getInstanceProfile(badInstanceId);
        System.out.println("The association Id value is " +
profileAssociationId);
        System.out.println("Replacing the profile for instance " + badInstanceId
                + " with a profile that contains bad credentials");
        autoScaler.replaceInstanceProfile(badInstanceId, badCredsProfileName,
profileAssociationId);

        System.out.println(
                """
                        Now, sending a GET request to the load balancer endpoint
returns either a recommendation or a static response,
                        depending on which instance is selected by the load
balancer.
                        """);

        demoChoices(loadBalancer);
```

```
        System.out.println("""
                Let's implement a deep health check. For this demo, a deep health
check tests whether
                the web service can access the DynamoDB table that it depends on
for recommendations. Note that
                the deep health check is only for ELB routing and not for Auto
Scaling instance health.
                This kind of deep health check is not recommended for Auto
Scaling instance health, because it
                risks accidental termination of all instances in the Auto Scaling
group when a dependent service fails.
                """);

        System.out.println("""
                By implementing deep health checks, the load balancer can detect
when one of the instances is failing
                and take that instance out of rotation.
                """);

        paramHelper.put(paramHelper.healthCheck, "deep");

        System.out.println("""
                Now, checking target health indicates that the instance with bad
credentials
                is unhealthy. Note that it might take a minute or two for the
load balancer to detect the unhealthy
                instance. Sending a GET request to the load balancer endpoint
always returns a recommendation, because
                the load balancer takes unhealthy instances out of its rotation.
                """);

        demoChoices(loadBalancer);

        System.out.println(
                """
                        Because the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy
                        instance is to terminate it and let the auto scaler start
a new instance to replace it.
                        """);
        autoScaler.terminateInstance(badInstanceId);

        System.out.println("""
```

```
                Even while the instance is terminating and the new instance is
starting, sending a GET
                request to the web service continues to get a successful
recommendation response because
                the load balancer routes requests to the healthy instances. After
the replacement instance
                starts and reports as healthy, it is included in the load
balancing rotation.
                Note that terminating and replacing an instance typically takes
several minutes, during which time you
                can see the changing health check status until the new instance
is running and healthy.
                """);

        demoChoices(loadBalancer);
        System.out.println(
                "If the recommendation service fails now, deep health checks mean
all instances report as unhealthy.");
        paramHelper.put(paramHelper.tableName, "this-is-not-a-table");

        demoChoices(loadBalancer);
        paramHelper.reset();
    }

    public static void demoChoices(LoadBalancer loadBalancer) throws IOException,
InterruptedException {
        String[] actions = {
                "Send a GET request to the load balancer endpoint.",
                "Check the health of load balancer targets.",
                "Go to the next part of the demo."
        };
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("-".repeat(88));
            System.out.println("See the current state of the service by selecting
one of the following choices:");
            for (int i = 0; i < actions.length; i++) {
                System.out.println(i + ": " + actions[i]);
            }

            try {
                System.out.print("\nWhich action would you like to take? ");
                int choice = scanner.nextInt();
```

```
                System.out.println("-".repeat(88));

                switch (choice) {
                    case 0 -> {
                        System.out.println("Request:\n");
                        System.out.println("GET http://" +
loadBalancer.getEndpoint(lbName));
                        CloseableHttpClient httpClient =
HttpClients.createDefault();

                        // Create an HTTP GET request to the ELB.
                        HttpGet httpGet = new HttpGet("http://" +
loadBalancer.getEndpoint(lbName));

                        // Execute the request and get the response.
                        HttpResponse response = httpClient.execute(httpGet);
                        int statusCode =
response.getStatusLine().getStatusCode();
                        System.out.println("HTTP Status Code: " + statusCode);

                        // Display the JSON response
                        BufferedReader reader = new BufferedReader(
                                new
InputStreamReader(response.getEntity().getContent()));
                        StringBuilder jsonResponse = new StringBuilder();
                        String line;
                        while ((line = reader.readLine()) != null) {
                            jsonResponse.append(line);
                        }
                        reader.close();

                        // Print the formatted JSON response.
                        System.out.println("Full Response:\n");
                        System.out.println(jsonResponse.toString());

                        // Close the HTTP client.
                        httpClient.close();

                    }
                    case 1 -> {
                        System.out.println("\nChecking the health of load
balancer targets:\n");
                        List<TargetHealthDescription> health =
loadBalancer.checkTargetHealth(targetGroupName);
```

```
                        for (TargetHealthDescription target : health) {
                            System.out.printf("\tTarget %s on port %d is %s%n",
 target.target().id(),
                                    target.target().port(),
 target.targetHealth().stateAsString());
                        }
                        System.out.println("""
                                Note that it can take a minute or two for the
 health check to update
                                after changes are made.
                                """);
                    }
                    case 2 -> {
                        System.out.println("\nOkay, let's move on.");
                        System.out.println("-".repeat(88));
                        return; // Exit the method when choice is 2
                    }
                    default -> System.out.println("You must choose a value
 between 0-2. Please select again.");
                }

            } catch (java.util.InputMismatchException e) {
                System.out.println("Invalid input. Please select again.");
                scanner.nextLine(); // Clear the input buffer.
            }
        }
    }

    public static String readFileAsString(String filePath) throws IOException {
        byte[] bytes = Files.readAllBytes(Paths.get(filePath));
        return new String(bytes);
    }
}
```

Create a class that wraps Auto Scaling and Amazon EC2 actions.

```
public class AutoScaler {

    private static Ec2Client ec2Client;
    private static AutoScalingClient autoScalingClient;
    private static IamClient iamClient;
```

```
    private static SsmClient ssmClient;

    private IamClient getIAMClient() {
        if (iamClient == null) {
            iamClient = IamClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return iamClient;
    }

    private SsmClient getSSMClient() {
        if (ssmClient == null) {
            ssmClient = SsmClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return ssmClient;
    }

    private Ec2Client getEc2Client() {
        if (ec2Client == null) {
            ec2Client = Ec2Client.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return ec2Client;
    }

    private AutoScalingClient getAutoScalingClient() {
        if (autoScalingClient == null) {
            autoScalingClient = AutoScalingClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return autoScalingClient;
    }

    /**
     * Terminates and instances in an EC2 Auto Scaling group. After an instance
is
     * terminated, it can no longer be accessed.
     */
    public void terminateInstance(String instanceId) {
```

```
        TerminateInstanceInAutoScalingGroupRequest terminateInstanceIRequest =
TerminateInstanceInAutoScalingGroupRequest
                .builder()
                .instanceId(instanceId)
                .shouldDecrementDesiredCapacity(false)
                .build();


getAutoScalingClient().terminateInstanceInAutoScalingGroup(terminateInstanceIRequest);
        System.out.format("Terminated instance %s.", instanceId);
    }

    /**
     * Replaces the profile associated with a running instance. After the profile
is
     * replaced, the instance is rebooted to ensure that it uses the new profile.
     * When
     * the instance is ready, Systems Manager is used to restart the Python web
     * server.
     */
    public void replaceInstanceProfile(String instanceId, String
newInstanceProfileName, String profileAssociationId)
            throws InterruptedException {
        // Create an IAM instance profile specification.
        software.amazon.awssdk.services.ec2.model.IamInstanceProfileSpecification
iamInstanceProfile =
software.amazon.awssdk.services.ec2.model.IamInstanceProfileSpecification
                .builder()
                .name(newInstanceProfileName) // Make sure
'newInstanceProfileName' is a valid IAM Instance Profile
                                              // name.
                .build();

        // Replace the IAM instance profile association for the EC2 instance.
        ReplaceIamInstanceProfileAssociationRequest replaceRequest =
ReplaceIamInstanceProfileAssociationRequest
                .builder()
                .iamInstanceProfile(iamInstanceProfile)
                .associationId(profileAssociationId) // Make sure
'profileAssociationId' is a valid association ID.
                .build();

        try {
            getEc2Client().replaceIamInstanceProfileAssociation(replaceRequest);
```

```
            // Handle the response as needed.
        } catch (Ec2Exception e) {
            // Handle exceptions, log, or report the error.
            System.err.println("Error: " + e.getMessage());
        }
        System.out.format("Replaced instance profile for association %s with
profile %s.", profileAssociationId,
                newInstanceProfileName);
        TimeUnit.SECONDS.sleep(15);
        boolean instReady = false;
        int tries = 0;

        // Reboot after 60 seconds
        while (!instReady) {
            if (tries % 6 == 0) {
                getEc2Client().rebootInstances(RebootInstancesRequest.builder()
                        .instanceIds(instanceId)
                        .build());
                System.out.println("Rebooting instance " + instanceId + " and
waiting for it to be ready.");
            }
            tries++;
            try {
                TimeUnit.SECONDS.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            DescribeInstanceInformationResponse informationResponse =
getSSMClient().describeInstanceInformation();
            List<InstanceInformation> instanceInformationList =
informationResponse.instanceInformationList();
            for (InstanceInformation info : instanceInformationList) {
                if (info.instanceId().equals(instanceId)) {
                    instReady = true;
                    break;
                }
            }
        }

        SendCommandRequest sendCommandRequest = SendCommandRequest.builder()
                .instanceIds(instanceId)
                .documentName("AWS-RunShellScript")
                .parameters(Collections.singletonMap("commands",
```

```
                        Collections.singletonList("cd / && sudo python3 server.py
80")))
                .build();

        getSSMClient().sendCommand(sendCommandRequest);
        System.out.println("Restarted the Python web server on instance " +
instanceId + ".");
    }

    public void openInboundPort(String secGroupId, String port, String ipAddress)
{
        AuthorizeSecurityGroupIngressRequest ingressRequest =
AuthorizeSecurityGroupIngressRequest.builder()
                .groupName(secGroupId)
                .cidrIp(ipAddress)
                .fromPort(Integer.parseInt(port))
                .build();

        getEc2Client().authorizeSecurityGroupIngress(ingressRequest);
        System.out.format("Authorized ingress to %s on port %s from %s.",
secGroupId, port, ipAddress);
    }

    /**
     * Detaches a role from an instance profile, detaches policies from the role,
     * and deletes all the resources.
     */
    public void deleteInstanceProfile(String roleName, String profileName) {
        try {
            software.amazon.awssdk.services.iam.model.GetInstanceProfileRequest
getInstanceProfileRequest =
software.amazon.awssdk.services.iam.model.GetInstanceProfileRequest
                    .builder()
                    .instanceProfileName(profileName)
                    .build();

            GetInstanceProfileResponse response =
getIAMClient().getInstanceProfile(getInstanceProfileRequest);
            String name = response.instanceProfile().instanceProfileName();
            System.out.println(name);

            RemoveRoleFromInstanceProfileRequest profileRequest =
RemoveRoleFromInstanceProfileRequest.builder()
                    .instanceProfileName(profileName)
```

```
                    .roleName(roleName)
                    .build();

            getIAMClient().removeRoleFromInstanceProfile(profileRequest);
            DeleteInstanceProfileRequest deleteInstanceProfileRequest =
DeleteInstanceProfileRequest.builder()
                    .instanceProfileName(profileName)
                    .build();

            getIAMClient().deleteInstanceProfile(deleteInstanceProfileRequest);
            System.out.println("Deleted instance profile " + profileName);

            DeleteRoleRequest deleteRoleRequest = DeleteRoleRequest.builder()
                    .roleName(roleName)
                    .build();

            // List attached role policies.
            ListAttachedRolePoliciesResponse rolesResponse = getIAMClient()
                    .listAttachedRolePolicies(role -> role.roleName(roleName));
            List<AttachedPolicy> attachedPolicies =
rolesResponse.attachedPolicies();
            for (AttachedPolicy attachedPolicy : attachedPolicies) {
                DetachRolePolicyRequest request =
DetachRolePolicyRequest.builder()
                        .roleName(roleName)
                        .policyArn(attachedPolicy.policyArn())
                        .build();

                getIAMClient().detachRolePolicy(request);
                System.out.println("Detached and deleted policy " +
attachedPolicy.policyName());
            }

            getIAMClient().deleteRole(deleteRoleRequest);
            System.out.println("Instance profile and role deleted.");

        } catch (IamException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public void deleteTemplate(String templateName) {
```

```
        getEc2Client().deleteLaunchTemplate(name ->
name.launchTemplateName(templateName));
        System.out.format(templateName + " was deleted.");
    }

    public void deleteAutoScaleGroup(String groupName) {
        DeleteAutoScalingGroupRequest deleteAutoScalingGroupRequest =
DeleteAutoScalingGroupRequest.builder()
                .autoScalingGroupName(groupName)
                .forceDelete(true)
                .build();


getAutoScalingClient().deleteAutoScalingGroup(deleteAutoScalingGroupRequest);
        System.out.println(groupName + " was deleted.");
    }

    /*
     * Verify the default security group of the specified VPC allows ingress from
     * this
     * computer. This can be done by allowing ingress from this computer's IP
     * address. In some situations, such as connecting from a corporate network,
you
     * must instead specify a prefix list ID. You can also temporarily open the
port
     * to
     * any IP address while running this example. If you do, be sure to remove
     * public
     * access when you're done.
     *
     */
    public GroupInfo verifyInboundPort(String VPC, int port, String ipAddress) {
        boolean portIsOpen = false;
        GroupInfo groupInfo = new GroupInfo();
        try {
            Filter filter = Filter.builder()
                    .name("group-name")
                    .values("default")
                    .build();

            Filter filter1 = Filter.builder()
                    .name("vpc-id")
                    .values(VPC)
                    .build();
```

```
            DescribeSecurityGroupsRequest securityGroupsRequest =
DescribeSecurityGroupsRequest.builder()
                    .filters(filter, filter1)
                    .build();

            DescribeSecurityGroupsResponse securityGroupsResponse =
getEc2Client()
                    .describeSecurityGroups(securityGroupsRequest);
            String securityGroup =
securityGroupsResponse.securityGroups().get(0).groupName();
            groupInfo.setGroupName(securityGroup);

            for (SecurityGroup secGroup :
securityGroupsResponse.securityGroups()) {
                System.out.println("Found security group: " +
secGroup.groupId());

                for (IpPermission ipPermission : secGroup.ipPermissions()) {
                    if (ipPermission.fromPort() == port) {
                        System.out.println("Found inbound rule: " +
ipPermission);
                        for (IpRange ipRange : ipPermission.ipRanges()) {
                            String cidrIp = ipRange.cidrIp();
                            if (cidrIp.startsWith(ipAddress) ||
cidrIp.equals("0.0.0.0/0")) {
                                System.out.println(cidrIp + " is applicable");
                                portIsOpen = true;
                            }
                        }

                        if (!ipPermission.prefixListIds().isEmpty()) {
                            System.out.println("Prefix lList is applicable");
                            portIsOpen = true;
                        }

                        if (!portIsOpen) {
                            System.out
                                    .println("The inbound rule does not appear to
be open to either this computer's IP,"
                                            + " all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                        } else {
                            break;
```

```
                        }
                    }
                }
            }

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }

        groupInfo.setPortOpen(portIsOpen);
        return groupInfo;
    }

    /*
     * Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
     * Scaling group.
     * The target group specifies how the load balancer forward requests to the
     * instances
     * in the group.
     */
    public void attachLoadBalancerTargetGroup(String asGroupName, String
targetGroupARN) {
        try {
            AttachLoadBalancerTargetGroupsRequest targetGroupsRequest =
AttachLoadBalancerTargetGroupsRequest.builder()
                    .autoScalingGroupName(asGroupName)
                    .targetGroupARNs(targetGroupARN)
                    .build();


getAutoScalingClient().attachLoadBalancerTargetGroups(targetGroupsRequest);
            System.out.println("Attached load balancer to " + asGroupName);

        } catch (AutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    // Creates an EC2 Auto Scaling group with the specified size.
    public String[] createGroup(int groupSize, String templateName, String
autoScalingGroupName) {

        // Get availability zones.
```

```
software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesRequest
zonesRequest =
software.amazon.awssdk.services.ec2.model.DescribeAvailabilityZonesRequest
               .builder()
               .build();

       DescribeAvailabilityZonesResponse zonesResponse =
getEc2Client().describeAvailabilityZones(zonesRequest);
       List<String> availabilityZoneNames =
zonesResponse.availabilityZones().stream()

.map(software.amazon.awssdk.services.ec2.model.AvailabilityZone::zoneName)
               .collect(Collectors.toList());

       String availabilityZones = String.join(",", availabilityZoneNames);
       LaunchTemplateSpecification specification =
LaunchTemplateSpecification.builder()
               .launchTemplateName(templateName)
               .version("$Default")
               .build();

       String[] zones = availabilityZones.split(",");
       CreateAutoScalingGroupRequest groupRequest =
CreateAutoScalingGroupRequest.builder()
               .launchTemplate(specification)
               .availabilityZones(zones)
               .maxSize(groupSize)
               .minSize(groupSize)
               .autoScalingGroupName(autoScalingGroupName)
               .build();

       try {
           getAutoScalingClient().createAutoScalingGroup(groupRequest);

       } catch (AutoScalingException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
       System.out.println("Created an EC2 Auto Scaling group named " +
autoScalingGroupName);
       return zones;
   }
```

```
   public String getDefaultVPC() {
       // Define the filter.
       Filter defaultFilter = Filter.builder()
               .name("is-default")
               .values("true")
               .build();

       software.amazon.awssdk.services.ec2.model.DescribeVpcsRequest request =
software.amazon.awssdk.services.ec2.model.DescribeVpcsRequest
               .builder()
               .filters(defaultFilter)
               .build();

       DescribeVpcsResponse response = getEc2Client().describeVpcs(request);
       return response.vpcs().get(0).vpcId();
   }

   // Gets the default subnets in a VPC for a specified list of Availability
Zones.
   public List<Subnet> getSubnets(String vpcId, String[] availabilityZones) {
       List<Subnet> subnets = null;
       Filter vpcFilter = Filter.builder()
               .name("vpc-id")
               .values(vpcId)
               .build();

       Filter azFilter = Filter.builder()
               .name("availability-zone")
               .values(availabilityZones)
               .build();

       Filter defaultForAZ = Filter.builder()
               .name("default-for-az")
               .values("true")
               .build();

       DescribeSubnetsRequest request = DescribeSubnetsRequest.builder()
               .filters(vpcFilter, azFilter, defaultForAZ)
               .build();

       DescribeSubnetsResponse response =
getEc2Client().describeSubnets(request);
       subnets = response.subnets();
       return subnets;
```

```
    }

    // Gets data about the instances in the EC2 Auto Scaling group.
    public String getBadInstance(String groupName) {
        DescribeAutoScalingGroupsRequest request =
DescribeAutoScalingGroupsRequest.builder()
                .autoScalingGroupNames(groupName)
                .build();

        DescribeAutoScalingGroupsResponse response =
getAutoScalingClient().describeAutoScalingGroups(request);
        AutoScalingGroup autoScalingGroup = response.autoScalingGroups().get(0);
        List<String> instanceIds = autoScalingGroup.instances().stream()
                .map(instance -> instance.instanceId())
                .collect(Collectors.toList());

        String[] instanceIdArray = instanceIds.toArray(new String[0]);
        for (String instanceId : instanceIdArray) {
            System.out.println("Instance ID: " + instanceId);
            return instanceId;
        }
        return "";
    }

    // Gets data about the profile associated with an instance.
    public String getInstanceProfile(String instanceId) {
        Filter filter = Filter.builder()
                .name("instance-id")
                .values(instanceId)
                .build();

        DescribeIamInstanceProfileAssociationsRequest associationsRequest =
DescribeIamInstanceProfileAssociationsRequest
                .builder()
                .filters(filter)
                .build();

        DescribeIamInstanceProfileAssociationsResponse response = getEc2Client()
                .describeIamInstanceProfileAssociations(associationsRequest);
        return response.iamInstanceProfileAssociations().get(0).associationId();
    }

    public void deleteRolesPolicies(String policyName, String roleName, String
InstanceProfile) {
```

```
        ListPoliciesRequest listPoliciesRequest =
 ListPoliciesRequest.builder().build();
        ListPoliciesResponse listPoliciesResponse =
 getIAMClient().listPolicies(listPoliciesRequest);
        for (Policy policy : listPoliciesResponse.policies()) {
            if (policy.policyName().equals(policyName)) {
                // List the entities (users, groups, roles) that are attached to
 the policy.

 software.amazon.awssdk.services.iam.model.ListEntitiesForPolicyRequest
 listEntitiesRequest =
 software.amazon.awssdk.services.iam.model.ListEntitiesForPolicyRequest
                        .builder()
                        .policyArn(policy.arn())
                        .build();
                ListEntitiesForPolicyResponse listEntitiesResponse = iamClient
                        .listEntitiesForPolicy(listEntitiesRequest);
                if (!listEntitiesResponse.policyGroups().isEmpty() || !
listEntitiesResponse.policyUsers().isEmpty()
                        || !listEntitiesResponse.policyRoles().isEmpty()) {
                    // Detach the policy from any entities it is attached to.
                    DetachRolePolicyRequest detachPolicyRequest =
 DetachRolePolicyRequest.builder()
                            .policyArn(policy.arn())
                            .roleName(roleName) // Specify the name of the IAM
 role
                            .build();

                    getIAMClient().detachRolePolicy(detachPolicyRequest);
                    System.out.println("Policy detached from entities.");
                }

                // Now, you can delete the policy.
                DeletePolicyRequest deletePolicyRequest =
 DeletePolicyRequest.builder()
                        .policyArn(policy.arn())
                        .build();

                getIAMClient().deletePolicy(deletePolicyRequest);
                System.out.println("Policy deleted successfully.");
                break;
            }
        }
```

```
        // List the roles associated with the instance profile
        ListInstanceProfilesForRoleRequest listRolesRequest =
 ListInstanceProfilesForRoleRequest.builder()
                .roleName(roleName)
                .build();

        // Detach the roles from the instance profile
        ListInstanceProfilesForRoleResponse listRolesResponse =
 iamClient.listInstanceProfilesForRole(listRolesRequest);
        for (software.amazon.awssdk.services.iam.model.InstanceProfile profile :
 listRolesResponse.instanceProfiles()) {
            RemoveRoleFromInstanceProfileRequest removeRoleRequest =
 RemoveRoleFromInstanceProfileRequest.builder()
                    .instanceProfileName(InstanceProfile)
                    .roleName(roleName) // Remove the extra dot here
                    .build();

            getIAMClient().removeRoleFromInstanceProfile(removeRoleRequest);
            System.out.println("Role " + roleName + " removed from instance
 profile " + InstanceProfile);
        }

        // Delete the instance profile after removing all roles
        DeleteInstanceProfileRequest deleteInstanceProfileRequest =
 DeleteInstanceProfileRequest.builder()
                .instanceProfileName(InstanceProfile)
                .build();

        getIAMClient().deleteInstanceProfile(r ->
 r.instanceProfileName(InstanceProfile));
        System.out.println(InstanceProfile + " Deleted");
        System.out.println("All roles and policies are deleted.");
    }
}
```

Create a class that wraps Elastic Load Balancing actions.

```
public class LoadBalancer {
    public ElasticLoadBalancingV2Client elasticLoadBalancingV2Client;

    public ElasticLoadBalancingV2Client getLoadBalancerClient() {
        if (elasticLoadBalancingV2Client == null) {
```

```
            elasticLoadBalancingV2Client = ElasticLoadBalancingV2Client.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }

        return elasticLoadBalancingV2Client;
    }

    // Checks the health of the instances in the target group.
    public List<TargetHealthDescription> checkTargetHealth(String
targetGroupName) {
        DescribeTargetGroupsRequest targetGroupsRequest =
DescribeTargetGroupsRequest.builder()
                .names(targetGroupName)
                .build();

        DescribeTargetGroupsResponse tgResponse =
getLoadBalancerClient().describeTargetGroups(targetGroupsRequest);

        DescribeTargetHealthRequest healthRequest =
DescribeTargetHealthRequest.builder()

.targetGroupArn(tgResponse.targetGroups().get(0).targetGroupArn())
                .build();

        DescribeTargetHealthResponse healthResponse =
getLoadBalancerClient().describeTargetHealth(healthRequest);
        return healthResponse.targetHealthDescriptions();
    }

    // Gets the HTTP endpoint of the load balancer.
    public String getEndpoint(String lbName) {
        DescribeLoadBalancersResponse res = getLoadBalancerClient()
                .describeLoadBalancers(describe -> describe.names(lbName));
        return res.loadBalancers().get(0).dnsName();
    }

    // Deletes a load balancer.
    public void deleteLoadBalancer(String lbName) {
        try {
            // Use a waiter to delete the Load Balancer.
            DescribeLoadBalancersResponse res = getLoadBalancerClient()
                    .describeLoadBalancers(describe -> describe.names(lbName));
```

```
            ElasticLoadBalancingV2Waiter loadBalancerWaiter =
getLoadBalancerClient().waiter();
            DescribeLoadBalancersRequest request =
DescribeLoadBalancersRequest.builder()

.loadBalancerArns(res.loadBalancers().get(0).loadBalancerArn())
                    .build();

            getLoadBalancerClient().deleteLoadBalancer(
                    builder ->
builder.loadBalancerArn(res.loadBalancers().get(0).loadBalancerArn()));
            WaiterResponse<DescribeLoadBalancersResponse> waiterResponse =
loadBalancerWaiter
                    .waitUntilLoadBalancersDeleted(request);
            waiterResponse.matched().response().ifPresent(System.out::println);

        } catch (ElasticLoadBalancingV2Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
        System.out.println(lbName + " was deleted.");
    }

    // Deletes the target group.
    public void deleteTargetGroup(String targetGroupName) {
        try {
            DescribeTargetGroupsResponse res = getLoadBalancerClient()
                    .describeTargetGroups(describe ->
describe.names(targetGroupName));
            getLoadBalancerClient()
                    .deleteTargetGroup(builder ->
builder.targetGroupArn(res.targetGroups().get(0).targetGroupArn()));
        } catch (ElasticLoadBalancingV2Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
        System.out.println(targetGroupName + " was deleted.");
    }

    // Verify this computer can successfully send a GET request to the load
balancer
    // endpoint.
    public boolean verifyLoadBalancerEndpoint(String elbDnsName) throws
IOException, InterruptedException {
        boolean success = false;
        int retries = 3;
```

```
        CloseableHttpClient httpClient = HttpClients.createDefault();

        // Create an HTTP GET request to the ELB.
        HttpGet httpGet = new HttpGet("http://" + elbDnsName);
        try {
            while ((!success) && (retries > 0)) {
                // Execute the request and get the response.
                HttpResponse response = httpClient.execute(httpGet);
                int statusCode = response.getStatusLine().getStatusCode();
                System.out.println("HTTP Status Code: " + statusCode);
                if (statusCode == 200) {
                    success = true;
                } else {
                    retries--;
                    System.out.println("Got connection error from load balancer
endpoint, retrying...");
                    TimeUnit.SECONDS.sleep(15);
                }
            }

        } catch (org.apache.http.conn.HttpHostConnectException e) {
            System.out.println(e.getMessage());
        }

        System.out.println("Status.." + success);
        return success;
    }

    /*
     * Creates an Elastic Load Balancing target group. The target group specifies
     * how
     * the load balancer forward requests to instances in the group and how
instance
     * health is checked.
     */
    public String createTargetGroup(String protocol, int port, String vpcId,
String targetGroupName) {
        CreateTargetGroupRequest targetGroupRequest =
CreateTargetGroupRequest.builder()
                .healthCheckPath("/healthcheck")
                .healthCheckTimeoutSeconds(5)
                .port(port)
                .vpcId(vpcId)
                .name(targetGroupName)
```

```
                .protocol(protocol)
                .build();

        CreateTargetGroupResponse targetGroupResponse =
getLoadBalancerClient().createTargetGroup(targetGroupRequest);
        String targetGroupArn =
targetGroupResponse.targetGroups().get(0).targetGroupArn();
        String targetGroup =
targetGroupResponse.targetGroups().get(0).targetGroupName();
        System.out.println("The " + targetGroup + " was created with ARN" +
targetGroupArn);
        return targetGroupArn;
    }

    /*
     * Creates an Elastic Load Balancing load balancer that uses the specified
     * subnets
     * and forwards requests to the specified target group.
     */
    public String createLoadBalancer(List<Subnet> subnetIds, String
targetGroupARN, String lbName, int port,
            String protocol) {
        try {
            List<String> subnetIdStrings = subnetIds.stream()
                    .map(Subnet::subnetId)
                    .collect(Collectors.toList());

            CreateLoadBalancerRequest balancerRequest =
CreateLoadBalancerRequest.builder()
                    .subnets(subnetIdStrings)
                    .name(lbName)
                    .scheme("internet-facing")
                    .build();

            // Create and wait for the load balancer to become available.
            CreateLoadBalancerResponse lsResponse =
getLoadBalancerClient().createLoadBalancer(balancerRequest);
            String lbARN = lsResponse.loadBalancers().get(0).loadBalancerArn();

            ElasticLoadBalancingV2Waiter loadBalancerWaiter =
getLoadBalancerClient().waiter();
            DescribeLoadBalancersRequest request =
DescribeLoadBalancersRequest.builder()
                    .loadBalancerArns(lbARN)
```

```
                    .build();

            System.out.println("Waiting for Load Balancer " + lbName + " to
 become available.");
            WaiterResponse<DescribeLoadBalancersResponse> waiterResponse =
 loadBalancerWaiter
                    .waitUntilLoadBalancerAvailable(request);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("Load Balancer " + lbName + " is available.");

            // Get the DNS name (endpoint) of the load balancer.
            String lbDNSName = lsResponse.loadBalancers().get(0).dnsName();
            System.out.println("*** Load Balancer DNS Name: " + lbDNSName);

            // Create a listener for the load balance.
            Action action = Action.builder()
                    .targetGroupArn(targetGroupARN)
                    .type("forward")
                    .build();

            CreateListenerRequest listenerRequest =
 CreateListenerRequest.builder()

 .loadBalancerArn(lsResponse.loadBalancers().get(0).loadBalancerArn())
                    .defaultActions(action)
                    .port(port)
                    .protocol(protocol)
                    .build();

            getLoadBalancerClient().createListener(listenerRequest);
            System.out.println("Created listener to forward traffic from load
 balancer " + lbName + " to target group "
                    + targetGroupARN);

            // Return the load balancer DNS name.
            return lbDNSName;

        } catch (ElasticLoadBalancingV2Exception e) {
            e.printStackTrace();
        }
        return "";
    }
}
```

Create a class that uses DynamoDB to simulate a recommendation service.

```
public class Database {

    private static DynamoDbClient dynamoDbClient;

    public static DynamoDbClient getDynamoDbClient() {
        if (dynamoDbClient == null) {
            dynamoDbClient = DynamoDbClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return dynamoDbClient;
    }

    // Checks to see if the Amazon DynamoDB table exists.
    private boolean doesTableExist(String tableName) {
        try {
            // Describe the table and catch any exceptions.
            DescribeTableRequest describeTableRequest =
 DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            getDynamoDbClient().describeTable(describeTableRequest);
            System.out.println("Table '" + tableName + "' exists.");
            return true;

        } catch (ResourceNotFoundException e) {
            System.out.println("Table '" + tableName + "' does not exist.");
        } catch (DynamoDbException e) {
            System.err.println("Error checking table existence: " +
 e.getMessage());
        }
        return false;
    }

    /*
     * Creates a DynamoDB table to use a recommendation service. The table has a
     * hash key named 'MediaType' that defines the type of media recommended,
 such
```

```
     * as
     * Book or Movie, and a range key named 'ItemId' that, combined with the
     * MediaType,
     * forms a unique identifier for the recommended item.
     */
    public void createTable(String tableName, String fileName) throws IOException
{
        // First check to see if the table exists.
        boolean doesExist = doesTableExist(tableName);
        if (!doesExist) {
            DynamoDbWaiter dbWaiter = getDynamoDbClient().waiter();
            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                    .tableName(tableName)
                    .attributeDefinitions(
                            AttributeDefinition.builder()
                                    .attributeName("MediaType")
                                    .attributeType(ScalarAttributeType.S)
                                    .build(),
                            AttributeDefinition.builder()
                                    .attributeName("ItemId")
                                    .attributeType(ScalarAttributeType.N)
                                    .build())
                    .keySchema(
                            KeySchemaElement.builder()
                                    .attributeName("MediaType")
                                    .keyType(KeyType.HASH)
                                    .build(),
                            KeySchemaElement.builder()
                                    .attributeName("ItemId")
                                    .keyType(KeyType.RANGE)
                                    .build())
                    .provisionedThroughput(
                            ProvisionedThroughput.builder()
                                    .readCapacityUnits(5L)
                                    .writeCapacityUnits(5L)
                                    .build())
                    .build();

            getDynamoDbClient().createTable(createTableRequest);
            System.out.println("Creating table " + tableName + "...");

            // Wait until the Amazon DynamoDB table is created.
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
```

```
                    .build();

            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("Table " + tableName + " created.");

            // Add records to the table.
            populateTable(fileName, tableName);
        }
    }

    public void deleteTable(String tableName) {
        getDynamoDbClient().deleteTable(table -> table.tableName(tableName));
        System.out.println("Table " + tableName + " deleted.");
    }

    // Populates the table with data located in a JSON file using the DynamoDB
    // enhanced client.
    public void populateTable(String fileName, String tableName) throws
IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(getDynamoDbClient())
                .build();
        ObjectMapper objectMapper = new ObjectMapper();
        File jsonFile = new File(fileName);
        JsonNode rootNode = objectMapper.readTree(jsonFile);

        DynamoDbTable<Recommendation> mappedTable =
enhancedClient.table(tableName,
                TableSchema.fromBean(Recommendation.class));
        for (JsonNode currentNode : rootNode) {
            String mediaType = currentNode.path("MediaType").path("S").asText();
            int itemId = currentNode.path("ItemId").path("N").asInt();
            String title = currentNode.path("Title").path("S").asText();
            String creator = currentNode.path("Creator").path("S").asText();

            // Create a Recommendation object and set its properties.
            Recommendation rec = new Recommendation();
            rec.setMediaType(mediaType);
            rec.setItemId(itemId);
            rec.setTitle(title);
            rec.setCreator(creator);
```

```
            // Put the item into the DynamoDB table.
            mappedTable.putItem(rec); // Add the Recommendation to the list.
        }
        System.out.println("Added all records to the " + tableName);
    }
}
```

Create a class that wraps Systems Manager actions.

```
public class ParameterHelper {

    String tableName = "doc-example-resilient-architecture-table";
    String dyntable = "doc-example-recommendation-service";
    String failureResponse = "doc-example-resilient-architecture-failure-
response";
    String healthCheck = "doc-example-resilient-architecture-health-check";

    public void reset() {
        put(dyntable, tableName);
        put(failureResponse, "none");
        put(healthCheck, "shallow");
    }

    public void put(String name, String value) {
        SsmClient ssmClient = SsmClient.builder()
                .region(Region.US_EAST_1)
                .build();

        PutParameterRequest parameterRequest = PutParameterRequest.builder()
                .name(name)
                .value(value)
                .overwrite(true)
                .type("String")
                .build();

        ssmClient.putParameter(parameterRequest);
        System.out.printf("Setting demo parameter %s to '%s'.", name, value);
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.

- AttachLoadBalancerTargetGroups
- CreateAutoScalingGroup
- CreateInstanceProfile
- CreateLaunchTemplate
- CreateListener
- CreateLoadBalancer
- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
```

```
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes]
 [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}
```

Create steps to deploy all of the resources.

```
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
```

```
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
```

```
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
```

```
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item']
[] }}
     */
    const recommendations = JSON.parse(
      readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
      new BatchWriteItemCommand({
        RequestItems: {
          [NAMES.tableName]: recommendations.map((item) => ({
            PutRequest: { Item: item },
          })),
        },
      }),
    );
  }),
  new ScenarioOutput(
    "populatedTable",
    MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
```

```
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
```

```
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
```

```
      new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    state.instanceProfileArn = Arn;

    await waitUntilInstanceProfileExists(
      { client },
      { InstanceProfileName: NAMES.instanceProfileName },
    );
  }),
  new ScenarioOutput("createdInstanceProfile", (state) =>
    MESSAGES.createdInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
  ),
  new ScenarioOutput(
    "addingRoleToInstanceProfile",
    MESSAGES.addingRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioAction("addRoleToInstanceProfile", () => {
    const client = new IAMClient({});
    return client.send(
      new AddRoleToInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  }),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
```

```
      }),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      }),
    );
  }),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
  new ScenarioOutput(
    "creatingAutoScalingGroup",
    MESSAGES.creatingAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
```

```
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    const client = new EC2Client({});
    const { Subnets } = await client.send(
```

```
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
```

```
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({
        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
```

```
        Port: state.targetGroupPort,
        DefaultActions: [
          { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
      }),
    );
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
  }),
  new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
      "${LB_LISTENER_ARN}",
      state.loadBalancerListenerArn,
    ),
  ),
  new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
  ),
  new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new AttachLoadBalancerTargetGroupsCommand({
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        TargetGroupARNs: [state.targetGroupArn],
      }),
    );
  }),
  new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup}} state
     */
    async (state) => {
      const client = new EC2Client({});
```

```
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
```

```
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      return MESSAGES.noIpRules;
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-
ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    }
    return false;
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
```

```
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];
```

Create steps to run the demo.

```
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
```

```
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
```

```
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
```

```
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
```

```
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
```

```
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-
scaling').Instance }} state
     */
    async (state) => {
      await createSsmOnlyInstanceProfile();
      const autoScalingClient = new AutoScalingClient({});
      const { AutoScalingGroups } = await autoScalingClient.send(
        new DescribeAutoScalingGroupsCommand({
          AutoScalingGroupNames: [NAMES.autoScalingGroupName],
        }),
      );
      state.targetInstance = AutoScalingGroups[0].Instances[0];
      const ec2Client = new EC2Client({});
```

```
      const { IamInstanceProfileAssociations } = await ec2Client.send(
        new DescribeIamInstanceProfileAssociationsCommand({
          Filters: [
            { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
          ],
        }),
      );
      state.instanceProfileAssociationId =
        IamInstanceProfileAssociations[0].AssociationId;
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        ec2Client.send(
          new ReplaceIamInstanceProfileAssociationCommand({
            AssociationId: state.instanceProfileAssociationId,
            IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
          }),
        ),
      );

      await ec2Client.send(
        new RebootInstancesCommand({
          InstanceIds: [state.targetInstance.InstanceId],
        }),
      );

      const ssmClient = new SSMClient({});
      await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
        const { InstanceInformationList } = await ssmClient.send(
          new DescribeInstanceInformationCommand({}),
        );

        const instance = InstanceInformationList.find(
          (info) => info.InstanceId === state.targetInstance.InstanceId,
        );

        if (!instance) {
          throw new Error("Instance not found.");
        }
      });

      await ssmClient.send(
        new SendCommandCommand({
          InstanceIds: [state.targetInstance.InstanceId],
          DocumentName: "AWS-RunShellScript",
          Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
```

```
        }),
      );
    },
  ),
  new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation}} state
     */
    (state) =>
      MESSAGES.demoTestBadCredentials.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
  ),
  loadBalancerLoop,
  new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
```

```
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
```

```
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
```

```
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );
  await iamClient.send(
    new AddRoleToInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      RoleName: NAMES.ssmOnlyRoleName,
```

```
    }),
  );

  return InstanceProfile;
}
```

Create steps to destroy all of the resources.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
```

```
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
```

```
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  }),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
    return MESSAGES.deletedKeyPair.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          }),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  }),
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
```

```
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          RoleName: NAMES.instanceRoleName,
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
```

```
      state.removeRoleFromInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
      console.error(state.removeRoleFromInstanceProfileError);
      return MESSAGES.removeRoleFromInstanceProfileError
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }),
  new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
```

```
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }),
  new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
      await terminateGroupInstances(NAMES.autoScalingGroupName);
      await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
      });
    } catch (e) {
      state.deleteAutoScalingGroupError = e;
    }
  }),
  new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
      console.error(state.deleteAutoScalingGroupError);
      return MESSAGES.deleteAutoScalingGroupError.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      );
    }
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }),
  new ScenarioAction("deleteLaunchTemplate", async (state) => {
```

```
    const client = new EC2Client({});
    try {
      await client.send(
        new DeleteLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
        }),
      );
    } catch (e) {
      state.deleteLaunchTemplateError = e;
    }
  }),
  new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
      console.error(state.deleteLaunchTemplateError);
      return MESSAGES.deleteLaunchTemplateError.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    }
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }),
  new ScenarioAction("deleteLoadBalancer", async (state) => {
    try {
      const client = new ElasticLoadBalancingV2Client({});
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      await client.send(
        new DeleteLoadBalancerCommand({
          LoadBalancerArn: loadBalancer.LoadBalancerArn,
        }),
      );
      await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
        const lb = await findLoadBalancer(NAMES.loadBalancerName);
        if (lb) {
          throw new Error("Load balancer still exists.");
        }
      });
    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
```

```
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        ),
      );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  }),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
```

```
  }),
  new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
```

```
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
    return MESSAGES.detachedSsmOnlyCustomRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  }),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  }),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
    return MESSAGES.detachedSsmOnlyAWSRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  }),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
```

```
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  }),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${POLICY_NAME}",
      NAMES.ssmOnlyPolicyName,
    );
  }),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
```

```
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }),
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }),
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      } catch (e) {
        state.revokeSecurityGroupIngressError = e;
      }
    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
```

```
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    }
    console.log(err.name);
    throw err;
  }
}

/**
```

```
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile

- CreateLaunchTemplate
- CreateListener
- CreateLoadBalancer
- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run the interactive scenario at a command prompt.

```
class Runner:
    """
    Manages the deployment, demonstration, and destruction of resources for the
 resilient service.
    """

    def __init__(
        self,
        resource_path: str,
        recommendation: RecommendationService,
        autoscaler: AutoScalingWrapper,
        loadbalancer: ElasticLoadBalancerWrapper,
        param_helper: ParameterHelper,
    ):
        """
        Initializes the Runner class with the necessary parameters.

        :param resource_path: The path to resource files used by this example,
 such as IAM policies and instance scripts.
        :param recommendation: An instance of the RecommendationService class.
        :param autoscaler: An instance of the AutoScaler class.
        :param loadbalancer: An instance of the LoadBalancer class.
        :param param_helper: An instance of the ParameterHelper class.
        """
        self.resource_path = resource_path
        self.recommendation = recommendation
        self.autoscaler = autoscaler
        self.loadbalancer = loadbalancer
        self.param_helper = param_helper
        self.protocol = "HTTP"
        self.port = 80
        self.ssh_port = 22

        prefix = "doc-example-resilience"
        self.target_group_name = f"{prefix}-tg"
        self.load_balancer_name = f"{prefix}-lb"

    def deploy(self) -> None:
        """
        Deploys the resources required for the resilient service, including the
 DynamoDB table,
        EC2 instances, Auto Scaling group, and load balancer.
```

```
        """
        recommendations_path = f"{self.resource_path}/recommendations.json"
        startup_script = f"{self.resource_path}/server_startup_script.sh"
        instance_policy = f"{self.resource_path}/instance_policy.json"

        logging.info("Starting deployment of resources for the resilient
service.")

        logging.info(
            "Creating and populating DynamoDB table '%s'.",
            self.recommendation.table_name,
        )
        self.recommendation.create()
        self.recommendation.populate(recommendations_path)

        logging.info(
            "Creating an EC2 launch template with the startup script '%s'.",
            startup_script,
        )
        self.autoscaler.create_template(startup_script, instance_policy)

        logging.info(
            "Creating an EC2 Auto Scaling group across multiple Availability
Zones."
        )
        zones = self.autoscaler.create_autoscaling_group(3)

        logging.info("Creating variables that control the flow of the demo.")
        self.param_helper.reset()

        logging.info("Creating Elastic Load Balancing target group and load
balancer.")

        vpc = self.autoscaler.get_default_vpc()
        subnets = self.autoscaler.get_subnets(vpc["VpcId"], zones)
        target_group = self.loadbalancer.create_target_group(
            self.target_group_name, self.protocol, self.port, vpc["VpcId"]
        )
        self.loadbalancer.create_load_balancer(
            self.load_balancer_name, [subnet["SubnetId"] for subnet in subnets]
        )
        self.loadbalancer.create_listener(self.load_balancer_name, target_group)

        self.autoscaler.attach_load_balancer_target_group(target_group)
```

```
        logging.info("Verifying access to the load balancer endpoint.")
        endpoint = self.loadbalancer.get_endpoint(self.load_balancer_name)
        lb_success = self.loadbalancer.verify_load_balancer_endpoint(endpoint)
        current_ip_address = requests.get("http://
checkip.amazonaws.com").text.strip()

        if not lb_success:
            logging.warning(
                "Couldn't connect to the load balancer. Verifying that the port
 is open..."
            )
            sec_group, port_is_open = self.autoscaler.verify_inbound_port(
                vpc, self.port, current_ip_address
            )
            sec_group, ssh_port_is_open = self.autoscaler.verify_inbound_port(
                vpc, self.ssh_port, current_ip_address
            )
            if not port_is_open:
                logging.warning(
                    "The default security group for your VPC must allow access
 from this computer."
                )
                if q.ask(
                    f"Do you want to add a rule to security group
 {sec_group['GroupId']} to allow\n"
                    f"inbound traffic on port {self.port} from your computer's IP
 address of {current_ip_address}? (y/n) ",
                    q.is_yesno,
                ):
                    self.autoscaler.open_inbound_port(
                        sec_group["GroupId"], self.port, current_ip_address
                    )
            if not ssh_port_is_open:
                if q.ask(
                    f"Do you want to add a rule to security group
 {sec_group['GroupId']} to allow\n"
                    f"inbound SSH traffic on port {self.ssh_port} for debugging
 from your computer's IP address of {current_ip_address}? (y/n) ",
                    q.is_yesno,
                ):
                    self.autoscaler.open_inbound_port(
                        sec_group["GroupId"], self.ssh_port, current_ip_address
                    )
```

```
            lb_success =
self.loadbalancer.verify_load_balancer_endpoint(endpoint)

        if lb_success:
            logging.info(
                "Load balancer is ready. Access it at: http://%s",
current_ip_address
            )
        else:
            logging.error(
                "Couldn't get a successful response from the load balancer
endpoint. Please verify your VPC and security group settings."
            )

    def demo_choices(self) -> None:
        """
        Presents choices for interacting with the deployed service, such as
sending requests to
        the load balancer or checking the health of the targets.
        """
        actions = [
            "Send a GET request to the load balancer endpoint.",
            "Check the health of load balancer targets.",
            "Go to the next part of the demo.",
        ]
        choice = 0
        while choice != 2:
            logging.info("Choose an action to interact with the service.")
            choice = q.choose("Which action would you like to take? ", actions)
            if choice == 0:
                logging.info("Sending a GET request to the load balancer
endpoint.")
                endpoint =
self.loadbalancer.get_endpoint(self.load_balancer_name)
                logging.info("GET http://%s", endpoint)
                response = requests.get(f"http://{endpoint}")
                logging.info("Response: %s", response.status_code)
                if response.headers.get("content-type") == "application/json":
                    pp(response.json())
            elif choice == 1:
                logging.info("Checking the health of load balancer targets.")
                health =
self.loadbalancer.check_target_health(self.target_group_name)
                for target in health:
```

```
                    state = target["TargetHealth"]["State"]
                    logging.info(
                        "Target %s on port %d is %s",
                        target["Target"]["Id"],
                        target["Target"]["Port"],
                        state,
                    )
                    if state != "healthy":
                        logging.warning(
                            "%s: %s",
                            target["TargetHealth"]["Reason"],
                            target["TargetHealth"]["Description"],
                        )
                logging.info(
                    "Note that it can take a minute or two for the health check
to update."
                )
            elif choice == 2:
                logging.info("Proceeding to the next part of the demo.")

    def demo(self) -> None:
        """
        Runs the demonstration, showing how the service responds to different
failure scenarios
        and how a resilient architecture can keep the service running.
        """
        ssm_only_policy = f"{self.resource_path}/ssm_only_policy.json"

        logging.info("Resetting parameters to starting values for the demo.")
        self.param_helper.reset()

        logging.info(
            "Starting demonstration of the service's resilience under various
failure conditions."
        )
        self.demo_choices()

        logging.info(
            "Simulating failure by changing the Systems Manager parameter to a
non-existent table."
        )
        self.param_helper.put(self.param_helper.table, "this-is-not-a-table")
        logging.info("Sending GET requests will now return failure codes.")
        self.demo_choices()
```

```
        logging.info("Switching to static response mode to mitigate failure.")
        self.param_helper.put(self.param_helper.failure_response, "static")
        logging.info("Sending GET requests will now return static responses.")
        self.demo_choices()

        logging.info("Restoring normal operation of the recommendation service.")
        self.param_helper.put(self.param_helper.table,
self.recommendation.table_name)

        logging.info(
            "Introducing a failure by assigning bad credentials to one of the
instances."
        )
        self.autoscaler.create_instance_profile(
            ssm_only_policy,
            self.autoscaler.bad_creds_policy_name,
            self.autoscaler.bad_creds_role_name,
            self.autoscaler.bad_creds_profile_name,
            ["AmazonSSMManagedInstanceCore"],
        )
        instances = self.autoscaler.get_instances()
        bad_instance_id = instances[0]
        instance_profile = self.autoscaler.get_instance_profile(bad_instance_id)
        logging.info(
            "Replacing instance profile with bad credentials for instance %s.",
            bad_instance_id,
        )
        self.autoscaler.replace_instance_profile(
            bad_instance_id,
            self.autoscaler.bad_creds_profile_name,
            instance_profile["AssociationId"],
        )
        logging.info(
            "Sending GET requests may return either a valid recommendation or a
static response."
        )
        self.demo_choices()

        logging.info("Implementing deep health checks to detect unhealthy
instances.")
        self.param_helper.put(self.param_helper.health_check, "deep")
        logging.info("Checking the health of the load balancer targets.")
        self.demo_choices()
```

```
        logging.info(
            "Terminating the unhealthy instance to let the auto scaler replace
it."
        )
        self.autoscaler.terminate_instance(bad_instance_id)
        logging.info("The service remains resilient during instance
replacement.")
        self.demo_choices()

        logging.info("Simulating a complete failure of the recommendation
service.")
        self.param_helper.put(self.param_helper.table, "this-is-not-a-table")
        logging.info(
            "All instances will report as unhealthy, but the service will still
return static responses."
        )
        self.demo_choices()
        self.param_helper.reset()

    def destroy(self, automation=False) -> None:
        """
        Destroys all resources created for the demo, including the load balancer,
Auto Scaling group,
        EC2 instances, and DynamoDB table.
        """
        logging.info(
            "This concludes the demo. Preparing to clean up all AWS resources
created during the demo."
        )
        if automation:
            cleanup = True
        else:
            cleanup = q.ask(
                "Do you want to clean up all demo resources? (y/n) ", q.is_yesno
            )

        if cleanup:
            logging.info("Deleting load balancer and related resources.")
            self.loadbalancer.delete_load_balancer(self.load_balancer_name)
            self.loadbalancer.delete_target_group(self.target_group_name)
            self.autoscaler.delete_autoscaling_group(self.autoscaler.group_name)
            self.autoscaler.delete_key_pair()
            self.autoscaler.delete_template()
```

```
            self.autoscaler.delete_instance_profile(
                self.autoscaler.bad_creds_profile_name,
                self.autoscaler.bad_creds_role_name,
            )
            logging.info("Deleting DynamoDB table and other resources.")
            self.recommendation.destroy()
        else:
            logging.warning(
                "Resources have not been deleted. Ensure you clean them up
 manually to avoid unexpected charges."
            )


def main() -> None:
    """
    Main function to parse arguments and run the appropriate actions for the
 demo.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--action",
        required=True,
        choices=["all", "deploy", "demo", "destroy"],
        help="The action to take for the demo. When 'all' is specified, resources
 are\n"
        "deployed, the demo is run, and resources are destroyed.",
    )
    parser.add_argument(
        "--resource_path",
        default="../../../scenarios/features/resilient_service/resources",
        help="The path to resource files used by this example, such as IAM
 policies and\n"
        "instance scripts.",
    )
    args = parser.parse_args()

    logging.info("Starting the Resilient Service demo.")

    prefix = "doc-example-resilience"

    # Service Clients
    ddb_client = boto3.client("dynamodb")
    elb_client = boto3.client("elbv2")
    autoscaling_client = boto3.client("autoscaling")
```

```
    ec2_client = boto3.client("ec2")
    ssm_client = boto3.client("ssm")
    iam_client = boto3.client("iam")

    # Wrapper instantiations
    recommendation = RecommendationService(
        "doc-example-recommendation-service", ddb_client
    )
    autoscaling_wrapper = AutoScalingWrapper(
        prefix,
        "t3.micro",
        "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
        autoscaling_client,
        ec2_client,
        ssm_client,
        iam_client,
    )
    elb_wrapper = ElasticLoadBalancerWrapper(elb_client)
    param_helper = ParameterHelper(recommendation.table_name, ssm_client)

    # Demo invocation
    runner = Runner(
        args.resource_path,
        recommendation,
        autoscaling_wrapper,
        elb_wrapper,
        param_helper,
    )
    actions = [args.action] if args.action != "all" else ["deploy", "demo",
 "destroy"]
    for action in actions:
        if action == "deploy":
            runner.deploy()
        elif action == "demo":
            runner.demo()
        elif action == "destroy":
            runner.destroy()

    logging.info("Demo completed successfully.")


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    main()
```

Create a class that wraps Auto Scaling and Amazon EC2 actions.

```
class AutoScalingWrapper:
    """
    Encapsulates Amazon EC2 Auto Scaling and EC2 management actions.
    """

    def __init__(
        self,
        resource_prefix: str,
        inst_type: str,
        ami_param: str,
        autoscaling_client: boto3.client,
        ec2_client: boto3.client,
        ssm_client: boto3.client,
        iam_client: boto3.client,
    ):
        """
        Initializes the AutoScaler class with the necessary parameters.

        :param resource_prefix: The prefix for naming AWS resources that are
 created by this class.
        :param inst_type: The type of EC2 instance to create, such as t3.micro.
        :param ami_param: The Systems Manager parameter used to look up the AMI
 that is created.
        :param autoscaling_client: A Boto3 EC2 Auto Scaling client.
        :param ec2_client: A Boto3 EC2 client.
        :param ssm_client: A Boto3 Systems Manager client.
        :param iam_client: A Boto3 IAM client.
        """
        self.inst_type = inst_type
        self.ami_param = ami_param
        self.autoscaling_client = autoscaling_client
        self.ec2_client = ec2_client
        self.ssm_client = ssm_client
        self.iam_client = iam_client
        sts_client = boto3.client("sts")
        self.account_id = sts_client.get_caller_identity()["Account"]

        self.key_pair_name = f"{resource_prefix}-key-pair"
        self.launch_template_name = f"{resource_prefix}-template-"
```

```
        self.group_name = f"{resource_prefix}-group"

        # Happy path
        self.instance_policy_name = f"{resource_prefix}-pol"
        self.instance_role_name = f"{resource_prefix}-role"
        self.instance_profile_name = f"{resource_prefix}-prof"

        # Failure mode
        self.bad_creds_policy_name = f"{resource_prefix}-bc-pol"
        self.bad_creds_role_name = f"{resource_prefix}-bc-role"
        self.bad_creds_profile_name = f"{resource_prefix}-bc-prof"


    def create_policy(self, policy_file: str, policy_name: str) -> str:
        """
        Creates a new IAM policy or retrieves the ARN of an existing policy.

        :param policy_file: The path to a JSON file that contains the policy
 definition.
        :param policy_name: The name to give the created policy.
        :return: The ARN of the created or existing policy.
        """
        with open(policy_file) as file:
            policy_doc = file.read()

        try:
            response = self.iam_client.create_policy(
                PolicyName=policy_name, PolicyDocument=policy_doc
            )
            policy_arn = response["Policy"]["Arn"]
            log.info(f"Policy '{policy_name}' created successfully. ARN:
 {policy_arn}")
            return policy_arn

        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityAlreadyExists":
                # If the policy already exists, get its ARN
                response = self.iam_client.get_policy(
                    PolicyArn=f"arn:aws:iam::{self.account_id}:policy/
{policy_name}"
                )
                policy_arn = response["Policy"]["Arn"]
                log.info(f"Policy '{policy_name}' already exists. ARN:
 {policy_arn}")
```

```
                return policy_arn
            log.error(f"Full error:\n\t{err}")

    def create_role(self, role_name: str, assume_role_doc: dict) -> str:
        """
        Creates a new IAM role or retrieves the ARN of an existing role.

        :param role_name: The name to give the created role.
        :param assume_role_doc: The assume role policy document that specifies
 which
                                entities can assume the role.
        :return: The ARN of the created or existing role.
        """
        try:
            response = self.iam_client.create_role(
                RoleName=role_name,
 AssumeRolePolicyDocument=json.dumps(assume_role_doc)
            )
            role_arn = response["Role"]["Arn"]
            log.info(f"Role '{role_name}' created successfully. ARN: {role_arn}")
            return role_arn

        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityAlreadyExists":
                # If the role already exists, get its ARN
                response = self.iam_client.get_role(RoleName=role_name)
                role_arn = response["Role"]["Arn"]
                log.info(f"Role '{role_name}' already exists. ARN: {role_arn}")
                return role_arn
            log.error(f"Full error:\n\t{err}")

    def attach_policy(
        self,
        role_name: str,
        policy_arn: str,
        aws_managed_policies: Tuple[str, ...] = (),
    ) -> None:
        """
        Attaches an IAM policy to a role and optionally attaches additional AWS-
managed policies.

        :param role_name: The name of the role to attach the policy to.
        :param policy_arn: The ARN of the policy to attach.
```

```
        :param aws_managed_policies: A tuple of AWS-managed policy names to
attach to the role.
        """
        try:
            self.iam_client.attach_role_policy(RoleName=role_name,
PolicyArn=policy_arn)
            for aws_policy in aws_managed_policies:
                self.iam_client.attach_role_policy(
                    RoleName=role_name,
                    PolicyArn=f"arn:aws:iam::aws:policy/{aws_policy}",
                )
            log.info(f"Attached policy {policy_arn} to role {role_name}.")
        except ClientError as err:
            log.error(f"Failed to attach policy {policy_arn} to role
{role_name}.")
            log.error(f"Full error:\n\t{err}")

    def create_instance_profile(
        self,
        policy_file: str,
        policy_name: str,
        role_name: str,
        profile_name: str,
        aws_managed_policies: Tuple[str, ...] = (),
    ) -> str:
        """
        Creates a policy, role, and profile that is associated with instances
created by
        this class. An instance's associated profile defines a role that is
assumed by the
        instance. The role has attached policies that specify the AWS permissions
granted to
        clients that run on the instance.

        :param policy_file: The name of a JSON file that contains the policy
definition to
                            create and attach to the role.
        :param policy_name: The name to give the created policy.
        :param role_name: The name to give the created role.
        :param profile_name: The name to the created profile.
        :param aws_managed_policies: Additional AWS-managed policies that are
attached to
                                     the role, such as
AmazonSSMManagedInstanceCore to grant
```

```
                                        use of Systems Manager to send commands to
the instance.
        :return: The ARN of the profile that is created.
        """
        assume_role_doc = {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {"Service": "ec2.amazonaws.com"},
                    "Action": "sts:AssumeRole",
                }
            ],
        }
        policy_arn = self.create_policy(policy_file, policy_name)
        self.create_role(role_name, assume_role_doc)
        self.attach_policy(role_name, policy_arn, aws_managed_policies)

        try:
            profile_response = self.iam_client.create_instance_profile(
                InstanceProfileName=profile_name
            )
            waiter = self.iam_client.get_waiter("instance_profile_exists")
            waiter.wait(InstanceProfileName=profile_name)
            time.sleep(10)  # wait a little longer
            profile_arn = profile_response["InstanceProfile"]["Arn"]
            self.iam_client.add_role_to_instance_profile(
                InstanceProfileName=profile_name, RoleName=role_name
            )
            log.info("Created profile %s and added role %s.", profile_name,
role_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityAlreadyExists":
                prof_response = self.iam_client.get_instance_profile(
                    InstanceProfileName=profile_name
                )
                profile_arn = prof_response["InstanceProfile"]["Arn"]
                log.info(
                    "Instance profile %s already exists, nothing to do.",
profile_name
                )
            log.error(f"Full error:\n\t{err}")
        return profile_arn
```

```
   def get_instance_profile(self, instance_id: str) -> Dict[str, Any]:
       """
       Gets data about the profile associated with an instance.

       :param instance_id: The ID of the instance to look up.
       :return: The profile data.
       """
       try:
           response =
self.ec2_client.describe_iam_instance_profile_associations(
               Filters=[{"Name": "instance-id", "Values": [instance_id]}]
           )
           if not response["IamInstanceProfileAssociations"]:
               log.info(f"No instance profile found for instance
{instance_id}.")
           profile_data = response["IamInstanceProfileAssociations"][0]
           log.info(f"Retrieved instance profile for instance {instance_id}.")
           return profile_data
       except ClientError as err:
           log.error(
               f"Failed to retrieve instance profile for instance
{instance_id}."
           )
           error_code = err.response["Error"]["Code"]
           if error_code == "InvalidInstanceID.NotFound":
               log.error(f"The instance ID '{instance_id}' does not exist.")
           log.error(f"Full error:\n\t{err}")


   def replace_instance_profile(
       self,
       instance_id: str,
       new_instance_profile_name: str,
       profile_association_id: str,
   ) -> None:
       """
       Replaces the profile associated with a running instance. After the
profile is
       replaced, the instance is rebooted to ensure that it uses the new
profile. When
       the instance is ready, Systems Manager is used to restart the Python web
server.
```

```
        :param instance_id: The ID of the instance to restart.
        :param new_instance_profile_name: The name of the new profile to
associate with
                                          the specified instance.
        :param profile_association_id: The ID of the existing profile association
for the
                                       instance.
        """
        try:
            self.ec2_client.replace_iam_instance_profile_association(
                IamInstanceProfile={"Name": new_instance_profile_name},
                AssociationId=profile_association_id,
            )
            log.info(
                "Replaced instance profile for association %s with profile %s.",
                profile_association_id,
                new_instance_profile_name,
            )
            time.sleep(5)

            self.ec2_client.reboot_instances(InstanceIds=[instance_id])
            log.info("Rebooting instance %s.", instance_id)
            waiter = self.ec2_client.get_waiter("instance_running")
            log.info("Waiting for instance %s to be running.", instance_id)
            waiter.wait(InstanceIds=[instance_id])
            log.info("Instance %s is now running.", instance_id)

            self.ssm_client.send_command(
                InstanceIds=[instance_id],
                DocumentName="AWS-RunShellScript",
                Parameters={"commands": ["cd / && sudo python3 server.py 80"]},
            )
            log.info(f"Restarted the Python web server on instance
'{instance_id}'.")
        except ClientError as err:
            log.error("Failed to replace instance profile.")
            error_code = err.response["Error"]["Code"]
            if error_code == "InvalidAssociationID.NotFound":
                log.error(
                    f"Association ID '{profile_association_id}' does not exist."
                    "Please check the association ID and try again."
                )
            if error_code == "InvalidInstanceId":
                log.error(
```

```
                    f"The specified instance ID '{instance_id}' does not exist or
is not available for SSM. "
                    f"Please verify the instance ID and try again."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_instance_profile(self, profile_name: str, role_name: str) -> None:
        """
        Detaches a role from an instance profile, detaches policies from the
role,
        and deletes all the resources.

        :param profile_name: The name of the profile to delete.
        :param role_name: The name of the role to delete.
        """
        try:
            self.iam_client.remove_role_from_instance_profile(
                InstanceProfileName=profile_name, RoleName=role_name
            )

self.iam_client.delete_instance_profile(InstanceProfileName=profile_name)
            log.info("Deleted instance profile %s.", profile_name)
            attached_policies = self.iam_client.list_attached_role_policies(
                RoleName=role_name
            )
            for pol in attached_policies["AttachedPolicies"]:
                self.iam_client.detach_role_policy(
                    RoleName=role_name, PolicyArn=pol["PolicyArn"]
                )
                if not pol["PolicyArn"].startswith("arn:aws:iam::aws"):
                    self.iam_client.delete_policy(PolicyArn=pol["PolicyArn"])
                log.info("Detached and deleted policy %s.", pol["PolicyName"])
            self.iam_client.delete_role(RoleName=role_name)
            log.info("Deleted role %s.", role_name)
        except ClientError as err:
            log.error(
                f"Couldn't delete instance profile {profile_name} or detach "
                f"policies and delete role {role_name}: {err}"
            )
            if err.response["Error"]["Code"] == "NoSuchEntity":
                log.info(
                    "Instance profile %s doesn't exist, nothing to do.",
profile_name
```

```
                )


    def create_key_pair(self, key_pair_name: str) -> None:
        """
        Creates a new key pair.

        :param key_pair_name: The name of the key pair to create.
        """
        try:
            response = self.ec2_client.create_key_pair(KeyName=key_pair_name)
            with open(f"{key_pair_name}.pem", "w") as file:
                file.write(response["KeyMaterial"])
            chmod(f"{key_pair_name}.pem", 0o600)
            log.info("Created key pair %s.", key_pair_name)
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to create key pair {key_pair_name}.")
            if error_code == "InvalidKeyPair.Duplicate":
                log.error(f"A key pair with the name '{key_pair_name}' already
exists.")
            log.error(f"Full error:\n\t{err}")


    def delete_key_pair(self) -> None:
        """
        Deletes a key pair.
        """
        try:
            self.ec2_client.delete_key_pair(KeyName=self.key_pair_name)
            remove(f"{self.key_pair_name}.pem")
            log.info("Deleted key pair %s.", self.key_pair_name)
        except ClientError as err:
            log.error(f"Couldn't delete key pair '{self.key_pair_name}'.")
            log.error(f"Full error:\n\t{err}")
        except FileNotFoundError as err:
            log.info("Key pair %s doesn't exist, nothing to do.",
self.key_pair_name)
            log.error(f"Full error:\n\t{err}")


    def create_template(
        self, server_startup_script_file: str, instance_policy_file: str
    ) -> Dict[str, Any]:
```

```
        """
        Creates an Amazon EC2 launch template to use with Amazon EC2 Auto
Scaling. The
        launch template specifies a Bash script in its user data field that runs
after
        the instance is started. This script installs Python packages and starts
a
        Python web server on the instance.

        :param server_startup_script_file: The path to a Bash script file that is
run
                                           when an instance starts.
        :param instance_policy_file: The path to a file that defines a
permissions policy
                                     to create and attach to the instance
profile.
        :return: Information about the newly created template.
        """
        template = {}
        try:
            # Create key pair and instance profile
            self.create_key_pair(self.key_pair_name)
            self.create_instance_profile(
                instance_policy_file,
                self.instance_policy_name,
                self.instance_role_name,
                self.instance_profile_name,
            )

            # Read the startup script
            with open(server_startup_script_file) as file:
                start_server_script = file.read()

            # Get the latest AMI ID
            ami_latest = self.ssm_client.get_parameter(Name=self.ami_param)
            ami_id = ami_latest["Parameter"]["Value"]

            # Create the launch template
            lt_response = self.ec2_client.create_launch_template(
                LaunchTemplateName=self.launch_template_name,
                LaunchTemplateData={
                    "InstanceType": self.inst_type,
                    "ImageId": ami_id,
                    "IamInstanceProfile": {"Name": self.instance_profile_name},
```

```
                    "UserData": base64.b64encode(
                        start_server_script.encode(encoding="utf-8")
                    ).decode(encoding="utf-8"),
                    "KeyName": self.key_pair_name,
                },
            )
            template = lt_response["LaunchTemplate"]
            log.info(
                f"Created launch template {self.launch_template_name} for AMI
{ami_id} on {self.inst_type}."
            )
        except ClientError as err:
            log.error(f"Failed to create launch template
{self.launch_template_name}.")
            error_code = err.response["Error"]["Code"]
            if error_code == "InvalidLaunchTemplateName.AlreadyExistsException":
                log.info(
                    f"Launch template {self.launch_template_name} already exists,
nothing to do."
                )
            log.error(f"Full error:\n\t{err}")
        return template


    def delete_template(self):
        """
        Deletes a launch template.
        """
        try:
            self.ec2_client.delete_launch_template(
                LaunchTemplateName=self.launch_template_name
            )
            self.delete_instance_profile(
                self.instance_profile_name, self.instance_role_name
            )
            log.info("Launch template %s deleted.", self.launch_template_name)
        except ClientError as err:
            if (
                err.response["Error"]["Code"]
                == "InvalidLaunchTemplateName.NotFoundException"
            ):
                log.info(
                    "Launch template %s does not exist, nothing to do.",
                    self.launch_template_name,
```

```
                )
            log.error(f"Full error:\n\t{err}")


    def get_availability_zones(self) -> List[str]:
        """
        Gets a list of Availability Zones in the AWS Region of the Amazon EC2
client.

        :return: The list of Availability Zones for the client Region.
        """
        try:
            response = self.ec2_client.describe_availability_zones()
            zones = [zone["ZoneName"] for zone in response["AvailabilityZones"]]
            log.info(f"Retrieved {len(zones)} availability zones: {zones}.")
        except ClientError as err:
            log.error("Failed to retrieve availability zones.")
            log.error(f"Full error:\n\t{err}")
        else:
            return zones


    def create_autoscaling_group(self, group_size: int) -> List[str]:
        """
        Creates an EC2 Auto Scaling group with the specified size.

        :param group_size: The number of instances to set for the minimum and
maximum in
                           the group.
        :return: The list of Availability Zones specified for the group.
        """
        try:
            zones = self.get_availability_zones()
            self.autoscaling_client.create_auto_scaling_group(
                AutoScalingGroupName=self.group_name,
                AvailabilityZones=zones,
                LaunchTemplate={
                    "LaunchTemplateName": self.launch_template_name,
                    "Version": "$Default",
                },
                MinSize=group_size,
                MaxSize=group_size,
            )
            log.info(
```

```
                f"Created EC2 Auto Scaling group {self.group_name} with
availability zones {zones}."
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            if error_code == "AlreadyExists":
                log.info(
                    f"EC2 Auto Scaling group {self.group_name} already exists,
nothing to do."
                )
            else:
                log.error(f"Failed to create EC2 Auto Scaling group
{self.group_name}.")
                log.error(f"Full error:\n\t{err}")
        else:
            return zones


    def get_instances(self) -> List[str]:
        """
        Gets data about the instances in the EC2 Auto Scaling group.

        :return: A list of instance IDs in the Auto Scaling group.
        """
        try:
            as_response = self.autoscaling_client.describe_auto_scaling_groups(
                AutoScalingGroupNames=[self.group_name]
            )
            instance_ids = [
                i["InstanceId"]
                for i in as_response["AutoScalingGroups"][0]["Instances"]
            ]
            log.info(
                f"Retrieved {len(instance_ids)} instances for Auto Scaling group
{self.group_name}."
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to retrieve instances for Auto Scaling group
{self.group_name}."
            )
            if error_code == "ResourceNotFound":
```

```
                log.error(f"The Auto Scaling group '{self.group_name}' does not
exist.")
            log.error(f"Full error:\n\t{err}")
        else:
            return instance_ids


    def terminate_instance(self, instance_id: str, decrementsetting=False) ->
None:
        """
        Terminates an instance in an EC2 Auto Scaling group. After an instance is
        terminated, it can no longer be accessed.

        :param instance_id: The ID of the instance to terminate.
        :param decrementsetting: If True, do not replace terminated instances.
        """
        try:
            self.autoscaling_client.terminate_instance_in_auto_scaling_group(
                InstanceId=instance_id,
                ShouldDecrementDesiredCapacity=decrementsetting,
            )
            log.info("Terminated instance %s.", instance_id)

            # Adding a waiter to ensure the instance is terminated
            waiter = self.ec2_client.get_waiter("instance_terminated")
            log.info("Waiting for instance %s to be terminated...", instance_id)
            waiter.wait(InstanceIds=[instance_id])
            log.info(
                f"Instance '{instance_id}' has been terminated and will be
replaced."
            )

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to terminate instance '{instance_id}'.")
            if error_code == "ScalingActivityInProgressFault":
                log.error(
                    "Scaling activity is currently in progress. "
                    "Wait for the scaling activity to complete before attempting
to terminate the instance again."
                )
            elif error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
```

```
                    "Ensure that no conflicting operations are being performed on
the resource."
                )
            log.error(f"Full error:\n\t{err}")

    def attach_load_balancer_target_group(
        self, lb_target_group: Dict[str, Any]
    ) -> None:
        """
        Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
        The target group specifies how the load balancer forwards requests to the
instances
        in the group.

        :param lb_target_group: Data about the ELB target group to attach.
        """
        try:
            self.autoscaling_client.attach_load_balancer_target_groups(
                AutoScalingGroupName=self.group_name,
                TargetGroupARNs=[lb_target_group["TargetGroupArn"]],
            )
            log.info(
                "Attached load balancer target group %s to auto scaling group
%s.",
                lb_target_group["TargetGroupName"],
                self.group_name,
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to attach load balancer target group
'{lb_target_group['TargetGroupName']}'."
            )
            if error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
                    "Ensure that no conflicting operations are being performed on
the resource."
                )
            elif error_code == "ServiceLinkedRoleFailure":
                log.error(
                    "The operation failed because the service-linked role is not
ready or does not exist. "
```

```
                    "Check that the service-linked role exists and is correctly
 configured."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_autoscaling_group(self, group_name: str) -> None:
        """
        Terminates all instances in the group, then deletes the EC2 Auto Scaling
 group.

        :param group_name: The name of the group to delete.
        """
        try:
            response = self.autoscaling_client.describe_auto_scaling_groups(
                AutoScalingGroupNames=[group_name]
            )
            groups = response.get("AutoScalingGroups", [])
            if len(groups) > 0:
                self.autoscaling_client.update_auto_scaling_group(
                    AutoScalingGroupName=group_name, MinSize=0
                )
                instance_ids = [inst["InstanceId"] for inst in groups[0]
["Instances"]]
                for inst_id in instance_ids:
                    self.terminate_instance(inst_id)

                # Wait for all instances to be terminated
                if instance_ids:
                    waiter = self.ec2_client.get_waiter("instance_terminated")
                    log.info("Waiting for all instances to be terminated...")
                    waiter.wait(InstanceIds=instance_ids)
                    log.info("All instances have been terminated.")
            else:
                log.info(f"No groups found named '{group_name}'! Nothing to do.")
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to delete Auto Scaling group '{group_name}'.")
            if error_code == "ScalingActivityInProgressFault":
                log.error(
                    "Scaling activity is currently in progress. "
                    "Wait for the scaling activity to complete before attempting
 to delete the group again."
                )
```

```
            elif error_code == "ResourceContentionFault":
                log.error(
                    "The request failed due to a resource contention issue. "
                    "Ensure that no conflicting operations are being performed on
 the group."
                )
            log.error(f"Full error:\n\t{err}")


    def get_default_vpc(self) -> Dict[str, Any]:
        """
        Gets the default VPC for the account.

        :return: Data about the default VPC.
        """
        try:
            response = self.ec2_client.describe_vpcs(
                Filters=[{"Name": "is-default", "Values": ["true"]}]
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error("Failed to retrieve the default VPC.")
            if error_code == "UnauthorizedOperation":
                log.error(
                    "You do not have the necessary permissions to describe VPCs.
 "
                    "Ensure that your AWS IAM user or role has the correct
 permissions."
                )
            elif error_code == "InvalidParameterValue":
                log.error(
                    "One or more parameters are invalid. Check the request
 parameters."
                )

            log.error(f"Full error:\n\t{err}")
        else:
            if "Vpcs" in response and response["Vpcs"]:
                log.info(f"Retrieved default VPC: {response['Vpcs'][0]
['VpcId']}")
                return response["Vpcs"][0]
            else:
                pass
```

```
   def verify_inbound_port(
       self, vpc: Dict[str, Any], port: int, ip_address: str
   ) -> Tuple[Dict[str, Any], bool]:
       """
       Verify the default security group of the specified VPC allows ingress
from this
       computer. This can be done by allowing ingress from this computer's IP
       address. In some situations, such as connecting from a corporate network,
you
       must instead specify a prefix list ID. You can also temporarily open the
port to
       any IP address while running this example. If you do, be sure to remove
public
       access when you're done.

       :param vpc: The VPC used by this example.
       :param port: The port to verify.
       :param ip_address: This computer's IP address.
       :return: The default security group of the specified VPC, and a value
that indicates
                whether the specified port is open.
       """
       try:
           response = self.ec2_client.describe_security_groups(
               Filters=[
                   {"Name": "group-name", "Values": ["default"]},
                   {"Name": "vpc-id", "Values": [vpc["VpcId"]]},
               ]
           )
           sec_group = response["SecurityGroups"][0]
           port_is_open = False
           log.info(f"Found default security group {sec_group['GroupId']}.")

           for ip_perm in sec_group["IpPermissions"]:
               if ip_perm.get("FromPort", 0) == port:
                   log.info(f"Found inbound rule: {ip_perm}")
                   for ip_range in ip_perm["IpRanges"]:
                       cidr = ip_range.get("CidrIp", "")
                       if cidr.startswith(ip_address) or cidr == "0.0.0.0/0":
                           port_is_open = True
                   if ip_perm["PrefixListIds"]:
                       port_is_open = True
                   if not port_is_open:
```

```
                        log.info(
                            f"The inbound rule does not appear to be open to
either this computer's IP "
                            f"address of {ip_address}, to all IP addresses
(0.0.0.0/0), or to a prefix list ID."
                        )
                    else:
                        break
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to verify inbound rule for port {port} for VPC
{vpc['VpcId']}."
            )
            if error_code == "InvalidVpcID.NotFound":
                log.error(
                    f"The specified VPC ID '{vpc['VpcId']}' does not exist.
Please check the VPC ID."
                )
            log.error(f"Full error:\n\t{err}")
        else:
            return sec_group, port_is_open


    def open_inbound_port(self, sec_group_id: str, port: int, ip_address: str) ->
None:
        """
        Add an ingress rule to the specified security group that allows access on
the
        specified port from the specified IP address.

        :param sec_group_id: The ID of the security group to modify.
        :param port: The port to open.
        :param ip_address: The IP address that is granted access.
        """
        try:
            self.ec2_client.authorize_security_group_ingress(
                GroupId=sec_group_id,
                CidrIp=f"{ip_address}/32",
                FromPort=port,
                ToPort=port,
                IpProtocol="tcp",
            )
            log.info(
```

```
                "Authorized ingress to %s on port %s from %s.",
                sec_group_id,
                port,
                ip_address,
            )
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to authorize ingress to security group '{sec_group_id}'
on port {port} from {ip_address}."
            )
            if error_code == "InvalidGroupId.Malformed":
                log.error(
                    "The security group ID is malformed. "
                    "Please verify that the security group ID is correct."
                )
            elif error_code == "InvalidPermission.Duplicate":
                log.error(
                    "The specified rule already exists in the security group. "
                    "Check the existing rules for this security group."
                )
            log.error(f"Full error:\n\t{err}")


    def get_subnets(self, vpc_id: str, zones: List[str] = None) -> List[Dict[str,
Any]]:
        """
        Gets the default subnets in a VPC for a specified list of Availability
Zones.

        :param vpc_id: The ID of the VPC to look up.
        :param zones: The list of Availability Zones to look up.
        :return: The list of subnets found.
        """
        # Ensure that 'zones' is a list, even if None is passed
        if zones is None:
            zones = []
        try:
            paginator = self.ec2_client.get_paginator("describe_subnets")
            page_iterator = paginator.paginate(
                Filters=[
                    {"Name": "vpc-id", "Values": [vpc_id]},
                    {"Name": "availability-zone", "Values": zones},
                    {"Name": "default-for-az", "Values": ["true"]},
```

```
                ]
            )

            subnets = []
            for page in page_iterator:
                subnets.extend(page["Subnets"])

            log.info("Found %s subnets for the specified zones.", len(subnets))
            return subnets
        except ClientError as err:
            log.error(
                f"Failed to retrieve subnets for VPC '{vpc_id}' in zones
 {zones}."
            )
            error_code = err.response["Error"]["Code"]
            if error_code == "InvalidVpcID.NotFound":
                log.error(
                    "The specified VPC ID does not exist. "
                    "Please check the VPC ID and try again."
                )
            # Add more error-specific handling as needed
            log.error(f"Full error:\n\t{err}")
```

Create a class that wraps Elastic Load Balancing actions.

```
class ElasticLoadBalancerWrapper:
    """Encapsulates Elastic Load Balancing (ELB) actions."""

    def __init__(self, elb_client: boto3.client):
        """
        Initializes the LoadBalancer class with the necessary parameters.
        """
        self.elb_client = elb_client


    def create_target_group(
        self, target_group_name: str, protocol: str, port: int, vpc_id: str
    ) -> Dict[str, Any]:
        """
```

```
        Creates an Elastic Load Balancing target group. The target group
specifies how
        the load balancer forwards requests to instances in the group and how
instance
        health is checked.

        To speed up this demo, the health check is configured with shortened
times and
        lower thresholds. In production, you might want to decrease the
sensitivity of
        your health checks to avoid unwanted failures.

        :param target_group_name: The name of the target group to create.
        :param protocol: The protocol to use to forward requests, such as 'HTTP'.
        :param port: The port to use to forward requests, such as 80.
        :param vpc_id: The ID of the VPC in which the load balancer exists.
        :return: Data about the newly created target group.
        """
        try:
            response = self.elb_client.create_target_group(
                Name=target_group_name,
                Protocol=protocol,
                Port=port,
                HealthCheckPath="/healthcheck",
                HealthCheckIntervalSeconds=10,
                HealthCheckTimeoutSeconds=5,
                HealthyThresholdCount=2,
                UnhealthyThresholdCount=2,
                VpcId=vpc_id,
            )
            target_group = response["TargetGroups"][0]
            log.info(f"Created load balancing target group
'{target_group_name}'.")
            return target_group
        except ClientError as err:
            log.error(
                f"Couldn't create load balancing target group
'{target_group_name}'."
            )
            error_code = err.response["Error"]["Code"]

            if error_code == "DuplicateTargetGroupName":
                log.error(
                    f"Target group name {target_group_name} already exists. "
```

```
                    "Check if the target group already exists."
                    "Consider using a different name or deleting the existing
target group if appropriate."
                )
            elif error_code == "TooManyTargetGroups":
                log.error(
                    "Too many target groups exist in the account. "
                    "Consider deleting unused target groups to create space for
new ones."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_target_group(self, target_group_name) -> None:
        """
        Deletes the target group.
        """
        try:
            # Describe the target group to get its ARN
            response =
self.elb_client.describe_target_groups(Names=[target_group_name])
            tg_arn = response["TargetGroups"][0]["TargetGroupArn"]

            # Delete the target group
            self.elb_client.delete_target_group(TargetGroupArn=tg_arn)
            log.info("Deleted load balancing target group %s.",
target_group_name)

            # Use a custom waiter to wait until the target group is no longer
available
            self.wait_for_target_group_deletion(self.elb_client, tg_arn)
            log.info("Target group %s successfully deleted.", target_group_name)

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to delete target group '{target_group_name}'.")
            if error_code == "TargetGroupNotFound":
                log.error(
                    "Load balancer target group either already deleted or never
existed. "
                    "Verify the name and check that the resource exists in the
AWS Console."
                )
            elif error_code == "ResourceInUseException":
```

```
                log.error(
                    "Target group still in use by another resource. "
                    "Ensure that the target group is no longer associated with
any load balancers or resources.",
                )
            log.error(f"Full error:\n\t{err}")

    def wait_for_target_group_deletion(
        self, elb_client, target_group_arn, max_attempts=10, delay=30
    ):
        for attempt in range(max_attempts):
            try:

elb_client.describe_target_groups(TargetGroupArns=[target_group_arn])
                print(
                    f"Attempt {attempt + 1}: Target group {target_group_arn}
still exists."
                )
            except ClientError as e:
                if e.response["Error"]["Code"] == "TargetGroupNotFound":
                    print(
                        f"Target group {target_group_arn} has been successfully
deleted."
                    )
                    return
                else:
                    raise
            time.sleep(delay)
        raise TimeoutError(
            f"Target group {target_group_arn} was not deleted after {max_attempts
* delay} seconds."
        )


    def create_load_balancer(
        self,
        load_balancer_name: str,
        subnet_ids: List[str],
    ) -> Dict[str, Any]:
        """
        Creates an Elastic Load Balancing load balancer that uses the specified
subnets
        and forwards requests to the specified target group.
```

```
        :param load_balancer_name: The name of the load balancer to create.
        :param subnet_ids: A list of subnets to associate with the load balancer.
        :return: Data about the newly created load balancer.
        """
        try:
            response = self.elb_client.create_load_balancer(
                Name=load_balancer_name, Subnets=subnet_ids
            )
            load_balancer = response["LoadBalancers"][0]
            log.info(f"Created load balancer '{load_balancer_name}'.")

            waiter = self.elb_client.get_waiter("load_balancer_available")
            log.info(
                f"Waiting for load balancer '{load_balancer_name}' to be
available..."
            )
            waiter.wait(Names=[load_balancer_name])
            log.info(f"Load balancer '{load_balancer_name}' is now available!")

        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to create load balancer '{load_balancer_name}'. Error
code: {error_code}, Message: {err.response['Error']['Message']}"
            )

            if error_code == "DuplicateLoadBalancerNameException":
                log.error(
                    f"A load balancer with the name '{load_balancer_name}'
already exists. "
                    "Load balancer names must be unique within the AWS region. "
                    "Please choose a different name and try again."
                )
            if error_code == "TooManyLoadBalancersException":
                log.error(
                    "The maximum number of load balancers has been reached in
this account and region. "
                    "You can delete unused load balancers or request an increase
in the service quota from AWS Support."
                )
            log.error(f"Full error:\n\t{err}")
        else:
            return load_balancer
```

```
   def create_listener(
       self,
       load_balancer_name: str,
       target_group: Dict[str, Any],
   ) -> Dict[str, Any]:
       """
       Creates a listener for the specified load balancer that forwards requests
to the
       specified target group.

       :param load_balancer_name: The name of the load balancer to create a
listener for.
       :param target_group: An existing target group that is added as a listener
to the
                            load balancer.
       :return: Data about the newly created listener.
       """
       try:
           # Retrieve the load balancer ARN
           load_balancer_response = self.elb_client.describe_load_balancers(
               Names=[load_balancer_name]
           )
           load_balancer_arn = load_balancer_response["LoadBalancers"][0][
               "LoadBalancerArn"
           ]

           # Create the listener
           response = self.elb_client.create_listener(
               LoadBalancerArn=load_balancer_arn,
               Protocol=target_group["Protocol"],
               Port=target_group["Port"],
               DefaultActions=[
                   {
                       "Type": "forward",
                       "TargetGroupArn": target_group["TargetGroupArn"],
                   }
               ],
           )
           log.info(
               f"Created listener to forward traffic from load balancer
'{load_balancer_name}' to target group '{target_group['TargetGroupName']}'."
           )
           return response["Listeners"][0]
```

```
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
                f"Failed to add a listener on '{load_balancer_name}' for target
group '{target_group['TargetGroupName']}'."
            )

            if error_code == "ListenerNotFoundException":
                log.error(
                    f"The listener could not be found for the load balancer
'{load_balancer_name}'. "
                    "Please check the load balancer name and target group
configuration."
                )
            if error_code == "InvalidConfigurationRequestException":
                log.error(
                    f"The configuration provided for the listener on load
balancer '{load_balancer_name}' is invalid. "
                    "Please review the provided protocol, port, and target group
settings."
                )
            log.error(f"Full error:\n\t{err}")


    def delete_load_balancer(self, load_balancer_name) -> None:
        """
        Deletes a load balancer.

        :param load_balancer_name: The name of the load balancer to delete.
        """
        try:
            response = self.elb_client.describe_load_balancers(
                Names=[load_balancer_name]
            )
            lb_arn = response["LoadBalancers"][0]["LoadBalancerArn"]
            self.elb_client.delete_load_balancer(LoadBalancerArn=lb_arn)
            log.info("Deleted load balancer %s.", load_balancer_name)
            waiter = self.elb_client.get_waiter("load_balancers_deleted")
            log.info("Waiting for load balancer to be deleted...")
            waiter.wait(Names=[load_balancer_name])
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(
```

```
                f"Couldn't delete load balancer '{load_balancer_name}'. Error
code: {error_code}, Message: {err.response['Error']['Message']}"
            )

            if error_code == "LoadBalancerNotFoundException":
                log.error(
                    f"The load balancer '{load_balancer_name}' does not exist. "
                    "Please check the name and try again."
                )
            log.error(f"Full error:\n\t{err}")


    def get_endpoint(self, load_balancer_name) -> str:
        """
        Gets the HTTP endpoint of the load balancer.

        :return: The endpoint.
        """
        try:
            response = self.elb_client.describe_load_balancers(
                Names=[load_balancer_name]
            )
            return response["LoadBalancers"][0]["DNSName"]
        except ClientError as err:
            log.error(
                f"Couldn't get the endpoint for load balancer
{load_balancer_name}"
            )
            error_code = err.response["Error"]["Code"]
            if error_code == "LoadBalancerNotFoundException":
                log.error(
                    "Verify load balancer name and ensure it exists in the AWS
console."
                )
            log.error(f"Full error:\n\t{err}")

    @staticmethod
    def verify_load_balancer_endpoint(endpoint) -> bool:
        """
        Verify this computer can successfully send a GET request to the load
balancer endpoint.

        :param endpoint: The endpoint to verify.
        :return: True if the GET request is successful, False otherwise.
```

```
        """
        retries = 3
        verified = False
        while not verified and retries > 0:
            try:
                lb_response = requests.get(f"http://{endpoint}")
                log.info(
                    "Got response %s from load balancer endpoint.",
                    lb_response.status_code,
                )
                if lb_response.status_code == 200:
                    verified = True
                else:
                    retries = 0
            except requests.exceptions.ConnectionError:
                log.info(
                    "Got connection error from load balancer endpoint,
retrying..."
                )
                retries -= 1
                time.sleep(10)
        return verified

    def check_target_health(self, target_group_name: str) -> List[Dict[str,
Any]]:
        """
        Checks the health of the instances in the target group.

        :return: The health status of the target group.
        """
        try:
            tg_response = self.elb_client.describe_target_groups(
                Names=[target_group_name]
            )
            health_response = self.elb_client.describe_target_health(
                TargetGroupArn=tg_response["TargetGroups"][0]["TargetGroupArn"]
            )
        except ClientError as err:
            log.error(f"Couldn't check health of {target_group_name} target(s).")
            error_code = err.response["Error"]["Code"]
            if error_code == "LoadBalancerNotFoundException":
                log.error(
                    "Load balancer associated with the target group was not
found. "
```

```
                    "Ensure the load balancer exists, is in the correct AWS
 region, and "
                    "that you have the necessary permissions to access it.",
                )
            elif error_code == "TargetGroupNotFoundException":
                log.error(
                    "Target group was not found. "
                    "Verify the target group name, check that it exists in the
 correct region, "
                    "and ensure it has not been deleted or created in a different
 account.",
                )
            log.error(f"Full error:\n\t{err}")
        else:
            return health_response["TargetHealthDescriptions"]
```

Create a class that uses DynamoDB to simulate a recommendation service.

```
class RecommendationService:
    """
    Encapsulates a DynamoDB table to use as a service that recommends books,
 movies,
    and songs.
    """

    def __init__(self, table_name: str, dynamodb_client: boto3.client):
        """
        Initializes the RecommendationService class with the necessary
 parameters.

        :param table_name: The name of the DynamoDB recommendations table.
        :param dynamodb_client: A Boto3 DynamoDB client.
        """
        self.table_name = table_name
        self.dynamodb_client = dynamodb_client

    def create(self) -> Dict[str, Any]:
        """
```

```
        Creates a DynamoDB table to use as a recommendation service. The table
has a
        hash key named 'MediaType' that defines the type of media recommended,
such as
        Book or Movie, and a range key named 'ItemId' that, combined with the
MediaType,
        forms a unique identifier for the recommended item.

        :return: Data about the newly created table.
        :raises RecommendationServiceError: If the table creation fails.
        """
        try:
            response = self.dynamodb_client.create_table(
                TableName=self.table_name,
                AttributeDefinitions=[
                    {"AttributeName": "MediaType", "AttributeType": "S"},
                    {"AttributeName": "ItemId", "AttributeType": "N"},
                ],
                KeySchema=[
                    {"AttributeName": "MediaType", "KeyType": "HASH"},
                    {"AttributeName": "ItemId", "KeyType": "RANGE"},
                ],
                ProvisionedThroughput={"ReadCapacityUnits": 5,
"WriteCapacityUnits": 5},
            )
            log.info("Creating table %s...", self.table_name)
            waiter = self.dynamodb_client.get_waiter("table_exists")
            waiter.wait(TableName=self.table_name)
            log.info("Table %s created.", self.table_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceInUseException":
                log.info("Table %s exists, nothing to be done.", self.table_name)
            else:
                raise RecommendationServiceError(
                    self.table_name, f"ClientError when creating table: {err}."
                )
        else:
            return response

    def populate(self, data_file: str) -> None:
        """
        Populates the recommendations table from a JSON file.

        :param data_file: The path to the data file.
```

```
        :raises RecommendationServiceError: If the table population fails.
        """
        try:
            with open(data_file) as data:
                items = json.load(data)
            batch = [{"PutRequest": {"Item": item}} for item in items]
            self.dynamodb_client.batch_write_item(RequestItems={self.table_name:
batch})
            log.info(
                "Populated table %s with items from %s.", self.table_name,
data_file
            )
        except ClientError as err:
            raise RecommendationServiceError(
                self.table_name, f"Couldn't populate table from {data_file}:
{err}"
            )

    def destroy(self) -> None:
        """
        Deletes the recommendations table.

        :raises RecommendationServiceError: If the table deletion fails.
        """
        try:
            self.dynamodb_client.delete_table(TableName=self.table_name)
            log.info("Deleting table %s...", self.table_name)
            waiter = self.dynamodb_client.get_waiter("table_not_exists")
            waiter.wait(TableName=self.table_name)
            log.info("Table %s deleted.", self.table_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                log.info("Table %s does not exist, nothing to do.",
self.table_name)
            else:
                raise RecommendationServiceError(
                    self.table_name, f"ClientError when deleting table: {err}."
                )
```

Create a class that wraps Systems Manager actions.

```
class ParameterHelper:
    """
    Encapsulates Systems Manager parameters. This example uses these parameters
 to drive
    the demonstration of resilient architecture, such as failure of a dependency
 or
    how the service responds to a health check.
    """

    table: str = "doc-example-resilient-architecture-table"
    failure_response: str = "doc-example-resilient-architecture-failure-response"
    health_check: str = "doc-example-resilient-architecture-health-check"

    def __init__(self, table_name: str, ssm_client: boto3.client):
        """
        Initializes the ParameterHelper class with the necessary parameters.

        :param table_name: The name of the DynamoDB table that is used as a
 recommendation
                           service.
        :param ssm_client: A Boto3 Systems Manager client.
        """
        self.ssm_client = ssm_client
        self.table_name = table_name

    def reset(self) -> None:
        """
        Resets the Systems Manager parameters to starting values for the demo.
        These are the name of the DynamoDB recommendation table, no response when
 a
        dependency fails, and shallow health checks.
        """
        self.put(self.table, self.table_name)
        self.put(self.failure_response, "none")
        self.put(self.health_check, "shallow")

    def put(self, name: str, value: str) -> None:
        """
        Sets the value of a named Systems Manager parameter.

        :param name: The name of the parameter.
        :param value: The new value of the parameter.
        :raises ParameterHelperError: If the parameter value cannot be set.
```

```
        """
        try:
            self.ssm_client.put_parameter(
                Name=name, Value=value, Overwrite=True, Type="String"
            )
            log.info("Setting parameter %s to '%s'.", name, value)
        except ClientError as err:
            error_code = err.response["Error"]["Code"]
            log.error(f"Failed to set parameter {name}.")
            if error_code == "ParameterLimitExceeded":
                log.error(
                    "The parameter limit has been exceeded. "
                    "Consider deleting unused parameters or request a limit
increase."
                )
            elif error_code == "ParameterAlreadyExists":
                log.error(
                    "The parameter already exists and overwrite is set to False.
"
                    "Use Overwrite=True to update the parameter."
                )
            log.error(f"Full error:\n\t{err}")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer

- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Configure container service connectivity

The following code example shows how to:

- Create the VPC infrastructure
- Set up logging
- Create the ECS cluster
- Configure IAM roles
- Create the service with Service Connect
- Verify the deployment
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# ECS Service Connect Tutorial Script v4 - Modified to use Default VPC
# This script creates an ECS cluster with Service Connect and deploys an nginx
 service
# Uses the default VPC to avoid VPC limits

set -e  # Exit on any error

# Configuration
SCRIPT_NAME="ECS Service Connect Tutorial"
LOG_FILE="ecs-service-connect-tutorial-v4-default-vpc.log"
REGION=${AWS_DEFAULT_REGION:-${AWS_REGION:-$(aws configure get region 2>/dev/
null)}}
if [ -z "$REGION" ]; then
    echo "ERROR: No AWS region configured."
    echo "Set one with: aws configure set region us-east-1"
    exit 1
fi
ENV_PREFIX="tutorial"
CLUSTER_NAME="${ENV_PREFIX}-cluster"
NAMESPACE_NAME="service-connect"

# Generate random suffix for unique resource names
RANDOM_SUFFIX=$(openssl rand -hex 6)

# Arrays to track created resources for cleanup
declare -a CREATED_RESOURCES=()

# Logging function
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
```

```
}

# Error handling function
handle_error() {
    log "ERROR: Script failed at line $1"
    log "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Set up error handling
trap 'handle_error $LINENO' ERR

# Function to add resource to tracking array
track_resource() {
    CREATED_RESOURCES+=("$1")
    log "Tracking resource: $1"
}

# Function to check if command output contains actual errors
check_for_errors() {
    local output="$1"
    local command_name="$2"

    # Check for specific AWS CLI error patterns, not just any occurrence of
 "error"
    if echo "$output" | grep -qi "An error occurred\|InvalidParameterException\|
AccessDenied\|ValidationException\|ResourceNotFoundException"; then
        log "ERROR in $command_name: $output"
        return 1
    fi
    return 0
}

# Function to get AWS account ID
get_account_id() {
    ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
    log "Using AWS Account ID: $ACCOUNT_ID"
}

# Function to wait for resources to be ready
wait_for_resource() {
    local resource_type="$1"
    local resource_id="$2"
```

```
    case "$resource_type" in
        "cluster")
            log "Waiting for cluster $resource_id to be active..."
            local attempt=1
            local max_attempts=30
            while [ $attempt -le $max_attempts ]; do
                local status=$(aws ecs describe-clusters --clusters
 "$resource_id" --query 'clusters[0].status' --output text)
                if [ "$status" = "ACTIVE" ]; then
                    log "Cluster is now active"
                    return 0
                fi
                log "Cluster status: $status (attempt $attempt/$max_attempts)"
                sleep 10
                ((attempt++))
            done
            log "ERROR: Cluster did not become active within expected time"
            return 1
            ;;
        "service")
            log "Waiting for service $resource_id to be stable..."
            aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
 "$resource_id"
            ;;
        "nat-gateway")
            log "Waiting for NAT Gateway $resource_id to be available..."
            aws ec2 wait nat-gateway-available --nat-gateway-ids "$resource_id"
            ;;
    esac
}

# Function to use default VPC infrastructure
setup_default_vpc_infrastructure() {
    log "Using default VPC infrastructure..."

    # Get default VPC
    VPC_ID=$(aws ec2 describe-vpcs --filters "Name=isDefault,Values=true" --query
 'Vpcs[0].VpcId' --output text)
    if [[ "$VPC_ID" == "None" || -z "$VPC_ID" ]]; then
        log "ERROR: No default VPC found. Please create a default VPC first."
        exit 1
    fi
    log "Using default VPC: $VPC_ID"
```

```
    # Get default subnets
    SUBNETS=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=$VPC_ID"
 "Name=default-for-az,Values=true" --query 'Subnets[].SubnetId' --output text)
    SUBNET_ARRAY=($SUBNETS)

    if [ ${#SUBNET_ARRAY[@]} -lt 2 ]; then
        log "ERROR: Need at least 2 subnets for ECS Service Connect. Found:
 ${#SUBNET_ARRAY[@]}"
        exit 1
    fi

    PUBLIC_SUBNET1=${SUBNET_ARRAY[0]}
    PUBLIC_SUBNET2=${SUBNET_ARRAY[1]}

    log "Using subnets: $PUBLIC_SUBNET1, $PUBLIC_SUBNET2"

    # Create security group for ECS tasks
    SG_OUTPUT=$(aws ec2 create-security-group \
        --group-name "${ENV_PREFIX}-ecs-sg-${RANDOM_SUFFIX}" \
        --description "Security group for ECS Service Connect tutorial" \
        --vpc-id "$VPC_ID" \
        --tag-specifications 'ResourceType=security-
group,Tags=[{Key=project,Value=doc-smith},{Key=tutorial,Value=amazon-ecs-service-
connect}]' 2>&1)
    check_for_errors "$SG_OUTPUT" "create-security-group"
    SECURITY_GROUP_ID=$(echo "$SG_OUTPUT" | grep -o '"GroupId": "[^"]*"' | cut -
d'"' -f4)
    track_resource "SG:$SECURITY_GROUP_ID"
    log "Created security group: $SECURITY_GROUP_ID"

    # Add inbound rules to security group
    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
        --protocol tcp \
        --port 80 \
        --cidr 0.0.0.0/0 >/dev/null 2>&1 || true

    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
        --protocol tcp \
        --port 443 \
        --cidr 0.0.0.0/0 >/dev/null 2>&1 || true
```

```
    log "Default VPC infrastructure setup completed"
}

# Function to create CloudWatch log groups
create_log_groups() {
    log "Creating CloudWatch log groups..."

    # Create log group for nginx container
    aws logs create-log-group --log-group-name "/ecs/service-connect-nginx"
 --tags project=doc-smith,tutorial=amazon-ecs-service-connect 2>&1 | grep -v
 "ResourceAlreadyExistsException" || {
        if [ ${PIPESTATUS[0]} -eq 0 ]; then
            log "Log group /ecs/service-connect-nginx created"
            track_resource "LOG_GROUP:/ecs/service-connect-nginx"
        else
            log "Log group /ecs/service-connect-nginx already exists"
        fi
    }

    # Create log group for service connect proxy
    aws logs create-log-group --log-group-name "/ecs/service-connect-proxy"
 --tags project=doc-smith,tutorial=amazon-ecs-service-connect 2>&1 | grep -v
 "ResourceAlreadyExistsException" || {
        if [ ${PIPESTATUS[0]} -eq 0 ]; then
            log "Log group /ecs/service-connect-proxy created"
            track_resource "LOG_GROUP:/ecs/service-connect-proxy"
        else
            log "Log group /ecs/service-connect-proxy already exists"
        fi
    }
}

# Function to create ECS cluster with Service Connect
create_ecs_cluster() {
    log "Creating ECS cluster with Service Connect..."

    CLUSTER_OUTPUT=$(aws ecs create-cluster \
        --cluster-name "$CLUSTER_NAME" \
        --service-connect-defaults namespace="$NAMESPACE_NAME" \
        --tags key=Environment,value=tutorial key=project,value=doc-smith
 key=tutorial,value=amazon-ecs-service-connect 2>&1)
    check_for_errors "$CLUSTER_OUTPUT" "create-cluster"

    track_resource "CLUSTER:$CLUSTER_NAME"
```

```
    log "Created ECS cluster: $CLUSTER_NAME"

    wait_for_resource "cluster" "$CLUSTER_NAME"

    # Track the Service Connect namespace that gets created
    # Wait a moment for the namespace to be created
    sleep 5
    NAMESPACE_ID=$(aws servicediscovery list-namespaces \
        --filters Name=TYPE,Values=HTTP \
        --query "Namespaces[?Name=='$NAMESPACE_NAME'].Id" --output text 2>/dev/
null || echo "")

    if [[ -n "$NAMESPACE_ID" && "$NAMESPACE_ID" != "None" ]]; then
        track_resource "NAMESPACE:$NAMESPACE_ID"
        log "Service Connect namespace created: $NAMESPACE_ID"
    fi
}

# Function to create IAM roles
create_iam_roles() {
    log "Creating IAM roles..."

    # Check if ecsTaskExecutionRole exists
    if aws iam get-role --role-name ecsTaskExecutionRole >/dev/null 2>&1; then
        log "IAM role ecsTaskExecutionRole exists"
    else
        log "Creating ecsTaskExecutionRole..."
        aws iam create-role \
            --role-name ecsTaskExecutionRole \
            --assume-role-policy-document '{
                "Version":"2012-10-17",
                "Statement": [{
                    "Effect": "Allow",
                    "Principal": {"Service": "ecs-tasks.amazonaws.com"},
                    "Action": "sts:AssumeRole"
                }]
            }' >/dev/null 2>&1
        aws iam attach-role-policy \
            --role-name ecsTaskExecutionRole \
            --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy >/dev/null 2>&1
        track_resource "ROLE:ecsTaskExecutionRole"
        aws iam tag-role --role-name ecsTaskExecutionRole --tags
 Key=project,Value=doc-smith Key=tutorial,Value=amazon-ecs-service-connect
```

```
        log "Created ecsTaskExecutionRole"
        sleep 10
    fi

    # Check if ecsTaskRole exists, create if not
    if aws iam get-role --role-name ecsTaskRole >/dev/null 2>&1; then
        log "IAM role ecsTaskRole exists"
    else
        log "IAM role ecsTaskRole does not exist, will create it"

        # Create trust policy for ECS tasks
        cat > /tmp/ecs-task-trust-policy.json << EOF
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF

        aws iam create-role \
            --role-name ecsTaskRole \
            --assume-role-policy-document file:///tmp/ecs-task-trust-policy.json
 >/dev/null

        track_resource "IAM_ROLE:ecsTaskRole"
        aws iam tag-role --role-name ecsTaskRole --tags Key=project,Value=doc-
smith Key=tutorial,Value=amazon-ecs-service-connect
        log "Created ecsTaskRole"

        # Wait for role to be available
        sleep 10
    fi
}

# Function to create task definition
create_task_definition() {
    log "Creating task definition..."
```

```
    # Create task definition JSON
    cat > /tmp/task-definition.json << EOF
{
    "family": "service-connect-nginx",
    "networkMode": "awsvpc",
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "256",
    "memory": "512",
    "executionRoleArn": "arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskExecutionRole",
    "taskRoleArn": "arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskRole",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/docker/library/nginx:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "protocol": "tcp",
                    "name": "nginx-port"
                }
            ],
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/service-connect-nginx",
                    "awslogs-region": "${REGION}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ]
}
EOF

    TASK_DEF_OUTPUT=$(aws ecs register-task-definition --cli-input-json file:///
tmp/task-definition.json 2>&1)
    check_for_errors "$TASK_DEF_OUTPUT" "register-task-definition"

    TASK_DEF_ARN=$(echo "$TASK_DEF_OUTPUT" | grep -o '"taskDefinitionArn":
 "[^"]*"' | cut -d'"' -f4)
    track_resource "TASK_DEF:service-connect-nginx"
    log "Created task definition: $TASK_DEF_ARN"
```

```
    # Clean up temporary file
    rm -f /tmp/task-definition.json
}

# Function to create ECS service with Service Connect
create_ecs_service() {
    log "Creating ECS service with Service Connect..."

    # Create service definition JSON
    cat > /tmp/service-definition.json << EOF
{
    "serviceName": "service-connect-nginx-service",
    "cluster": "${CLUSTER_NAME}",
    "taskDefinition": "service-connect-nginx",
    "desiredCount": 1,
    "launchType": "FARGATE",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": ["${PUBLIC_SUBNET1}", "${PUBLIC_SUBNET2}"],
            "securityGroups": ["${SECURITY_GROUP_ID}"],
            "assignPublicIp": "ENABLED"
        }
    },
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "${NAMESPACE_NAME}",
        "services": [
            {
                "portName": "nginx-port",
                "discoveryName": "nginx",
                "clientAliases": [
                    {
                        "port": 80,
                        "dnsName": "nginx"
                    }
                ]
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "${REGION}",
```

```
                "awslogs-stream-prefix": "ecs-service-connect"
            }
        }
    },
    "tags": [
        {
            "key": "Environment",
            "value": "tutorial"
        },
        {
            "key": "project",
            "value": "doc-smith"
        },
        {
            "key": "tutorial",
            "value": "amazon-ecs-service-connect"
        }
    ]
}
EOF

    SERVICE_OUTPUT=$(aws ecs create-service --cli-input-json file:///tmp/service-
definition.json 2>&1)
    check_for_errors "$SERVICE_OUTPUT" "create-service"

    track_resource "SERVICE:service-connect-nginx-service"
    log "Created ECS service: service-connect-nginx-service"

    wait_for_resource "service" "service-connect-nginx-service"

    # Clean up temporary file
    rm -f /tmp/service-definition.json
}

# Function to verify deployment
verify_deployment() {
    log "Verifying deployment..."

    # Check service status
    SERVICE_STATUS=$(aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].status' --output text)
    log "Service status: $SERVICE_STATUS"
```

```
    # Check running tasks
    RUNNING_COUNT=$(aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].runningCount' --output text)
    log "Running tasks: $RUNNING_COUNT"

    # Get task ARN
    TASK_ARN=$(aws ecs list-tasks \
        --cluster "$CLUSTER_NAME" \
        --service-name "service-connect-nginx-service" \
        --query 'taskArns[0]' --output text)

    if [[ "$TASK_ARN" != "None" && -n "$TASK_ARN" ]]; then
        log "Task ARN: $TASK_ARN"

        # Try to get task IP address
        TASK_IP=$(aws ecs describe-tasks \
            --cluster "$CLUSTER_NAME" \
            --tasks "$TASK_ARN" \
            --query 'tasks[0].attachments[0].details[?
name==`privateIPv4Address`].value' \
            --output text 2>/dev/null || echo "")

        if [[ -n "$TASK_IP" && "$TASK_IP" != "None" ]]; then
            log "Task IP address: $TASK_IP"
        else
            log "Could not retrieve task IP address"
        fi
    fi

    # Check Service Connect namespace
    NAMESPACE_STATUS=$(aws servicediscovery list-namespaces \
        --filters Name=TYPE,Values=HTTP \
        --query "Namespaces[?Name=='$NAMESPACE_NAME'].Id" --output text 2>/dev/
null || echo "")

    if [[ -n "$NAMESPACE_STATUS" && "$NAMESPACE_STATUS" != "None" ]]; then
        log "Service Connect namespace '$NAMESPACE_NAME' is active"
    else
        log "Service Connect namespace '$NAMESPACE_NAME' not found or not active"
    fi
```

```
    # Display Service Connect configuration
    log "Service Connect configuration:"
    aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].serviceConnectConfiguration' 2>/dev/null || true
}

# Function to display created resources
display_resources() {
    echo ""
    echo "============================================="
    echo "CREATED RESOURCES"
    echo "============================================="
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "- $resource"
    done
    echo "============================================="
    echo ""
}

# Function to cleanup resources
cleanup_resources() {
    log "Starting cleanup process..."

    # Delete resources in reverse order of creation
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        resource="${CREATED_RESOURCES[i]}"
        resource_type=$(echo "$resource" | cut -d':' -f1)
        resource_id=$(echo "$resource" | cut -d':' -f2)

        log "Cleaning up $resource_type: $resource_id"

        case "$resource_type" in
            "SERVICE")
                aws ecs update-service --cluster "$CLUSTER_NAME" --service
 "$resource_id" --desired-count 0 2>&1 | grep -qi "error" && log "Warning: Failed
 to scale down service $resource_id"
                aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
 "$resource_id" 2>/dev/null || true
                aws ecs delete-service --cluster "$CLUSTER_NAME" --service
 "$resource_id" --force 2>&1 | grep -qi "error" && log "Warning: Failed to delete
 service $resource_id"
                ;;
```

```
            "TASK_DEF")
                TASK_DEF_ARNS=$(aws ecs list-task-definitions --family-prefix
 "$resource_id" --query 'taskDefinitionArns' --output text 2>/dev/null)
                for arn in $TASK_DEF_ARNS; do
                    aws ecs deregister-task-definition --task-definition "$arn"
 >/dev/null 2>&1 || true
                done
                ;;
            "ROLE")
                aws iam detach-role-policy --role-name "$resource_id" --policy-
arn "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy" 2>/
dev/null || true
                aws iam delete-role --role-name "$resource_id" 2>&1 | grep -qi
 "error" && log "Warning: Failed to delete role $resource_id"
                ;;
            "IAM_ROLE")
                aws iam detach-role-policy --role-name "$resource_id" --policy-
arn "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy" 2>/
dev/null || true
                aws iam delete-role --role-name "$resource_id" 2>&1 | grep -qi
 "error" && log "Warning: Failed to delete role $resource_id"
                ;;
            "CLUSTER")
                aws ecs delete-cluster --cluster "$resource_id" 2>&1 | grep -qi
 "error" && log "Warning: Failed to delete cluster $resource_id"
                ;;
            "SG")
                for attempt in 1 2 3 4 5; do
                    if aws ec2 delete-security-group --group-id "$resource_id"
 2>/dev/null; then
                        break
                    fi
                    log "Security group $resource_id still has dependencies,
 retrying in 30s ($attempt/5)..."
                    sleep 30
                done
                ;;
            "LOG_GROUP")
                aws logs delete-log-group --log-group-name "$resource_id" 2>&1 |
 grep -qi "error" && log "Warning: Failed to delete log group $resource_id"
                ;;
            "NAMESPACE")
                # First, delete any services in the namespace
                NAMESPACE_SERVICES=$(aws servicediscovery list-services \
```

```
                    --filters Name=NAMESPACE_ID,Values="$resource_id" \
                    --query 'Services[].Id' --output text 2>/dev/null || echo "")

                if [[ -n "$NAMESPACE_SERVICES" && "$NAMESPACE_SERVICES" !=
 "None" ]]; then
                    for service_id in $NAMESPACE_SERVICES; do
                        aws servicediscovery delete-service --id "$service_id" >/
dev/null 2>&1 || true
                        sleep 2
                    done
                fi

                # Then delete the namespace
                aws servicediscovery delete-namespace --id "$resource_id" >/dev/
null 2>&1 || true
                ;;
        esac

        sleep 2  # Brief pause between deletions
    done

    # Clean up temporary files
    rm -f /tmp/ecs-task-trust-policy.json
    rm -f /tmp/task-definition.json
    rm -f /tmp/service-definition.json

    log "Cleanup completed"
}

# Main execution
main() {
    log "Starting $SCRIPT_NAME v4 (Default VPC)"
    log "Region: $REGION"
    log "Log file: $LOG_FILE"

    # Get AWS account ID
    get_account_id

    # Setup infrastructure using default VPC
    setup_default_vpc_infrastructure

    # Create CloudWatch log groups
    create_log_groups
```

```
    # Create ECS cluster
    create_ecs_cluster

    # Create IAM roles
    create_iam_roles

    # Create task definition
    create_task_definition

    # Create ECS service
    create_ecs_service

    # Verify deployment
    verify_deployment

    log "Tutorial completed successfully!"

    # Display created resources
    display_resources

    # Ask user if they want to clean up
    echo ""
    echo "============================================"
    echo "CLEANUP CONFIRMATION"
    echo "============================================"
    echo "Do you want to clean up all created resources? (y/n): "
    CLEANUP_CHOICE="y"

    if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
        cleanup_resources
        log "All resources have been cleaned up"
    else
        log "Resources left intact. You can clean them up later by running the
 cleanup function."
        echo ""
        echo "To clean up resources later, you can use the AWS CLI commands or
 the AWS Management Console."
        echo "Remember to delete resources in the correct order to avoid
 dependency issues."
    fi
}

# Make script executable and run
chmod +x "$0"
```

```
main "$@"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - AuthorizeSecurityGroupIngress
  - CreateCluster
  - CreateLogGroup
  - CreateRole
  - CreateSecurityGroup
  - CreateService
  - DeleteCluster
  - DeleteLogGroup
  - DeleteNamespace
  - DeleteRole
  - DeleteSecurityGroup
  - DeleteService (AWS Cloud Map)
  - DeleteService (Amazon ECS)
  - DeregisterTaskDefinition
  - DescribeClusters
  - DescribeServices
  - DescribeSubnets
  - DescribeTasks
  - DescribeVpcs
  - DetachRolePolicy
  - GetCallerIdentity
  - GetRole
  - ListNamespaces
  - ListServices
  - ListTaskDefinitions
  - ListTasks

- RegisterTaskDefinition
- UpdateService
- Wait (Amazon EC2)
- Wait (Amazon ECS)

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a container task for the serverless launch type

The following code example shows how to:

- Create the cluster
- Create a task definition
- Create the service
- Clean up

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon ECS Fargate Tutorial Script - Version 5
# This script creates an ECS cluster, task definition, and service using Fargate
 launch type
# Fixed version with proper resource dependency handling during cleanup
# Security improvements applied

set -e  # Exit on any error
```

```
# Initialize logging
LOG_FILE="ecs-fargate-tutorial-v5.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon ECS Fargate tutorial at $(date)"
echo "Log file: $LOG_FILE"

# Generate random identifier for unique resource names
RANDOM_ID=$(openssl rand -hex 6)
CLUSTER_NAME="fargate-cluster-${RANDOM_ID}"
SERVICE_NAME="fargate-service-${RANDOM_ID}"
TASK_FAMILY="sample-fargate-${RANDOM_ID}"
SECURITY_GROUP_NAME="ecs-fargate-sg-${RANDOM_ID}"

# Array to track created resources for cleanup
CREATED_RESOURCES=()

# Function to log and execute commands with input validation
execute_command() {
    local cmd="$1"
    local description="$2"

    # Validate that cmd is not empty
    if [[ -z "$cmd" ]]; then
        echo "ERROR: Command is empty"
        return 1
    fi

    echo ""
    echo "=========================================="
    echo "EXECUTING: $description"
    echo "COMMAND: $cmd"
    echo "=========================================="

    local output
    local exit_code
    set +e  # Temporarily disable exit on error
    output=$(eval "$cmd" 2>&1)
    exit_code=$?
    set -e  # Re-enable exit on error

    if [[ $exit_code -eq 0 ]]; then
        echo "SUCCESS: $description"
        echo "OUTPUT: $output"
```

```
        return 0
    else
        echo "FAILED: $description"
        echo "EXIT CODE: $exit_code"
        echo "OUTPUT: $output"
        return 1
    fi
}

# Function to check for actual AWS API errors in command output
check_for_aws_errors() {
    local output="$1"
    local description="$2"

    # Look for specific AWS error patterns, not just the word "error"
    if echo "$output" | grep -qi "An error occurred\|InvalidParameter\|
AccessDenied\|ResourceNotFound\|ValidationException"; then
        echo "AWS API ERROR detected in output for: $description"
        echo "Output: $output"
        return 1
    fi
    return 0
}

# Function to safely extract JSON values
safe_json_extract() {
    local json="$1"
    local key="$2"
    local value

    value=$(echo "$json" | grep -o "\"$key\": \"[^\"]*\"" | cut -d'"' -f4 2>/dev/
null || echo "")
    echo "$value"
}

# Function to wait for network interfaces to be cleaned up
wait_for_network_interfaces_cleanup() {
    local security_group_id="$1"
    local max_attempts=30
    local attempt=1

    # Validate security group ID format
    if [[ ! "$security_group_id" =~ ^sg-[a-z0-9]{8,17}$ ]]; then
        echo "ERROR: Invalid security group ID format: $security_group_id"
```

```
        return 1
    fi

    echo "Waiting for network interfaces to be cleaned up..."

    while [[ $attempt -le $max_attempts ]]; do
        echo "Attempt $attempt/$max_attempts: Checking for dependent network
 interfaces..."

        # Check if there are any network interfaces still using this security
 group
        local eni_count
        eni_count=$(aws ec2 describe-network-interfaces \
            --filters "Name=group-id,Values=$security_group_id" \
            --query "length(NetworkInterfaces)" \
            --output text 2>/dev/null || echo "0")

        if [[ "$eni_count" == "0" ]]; then
            echo "No network interfaces found using security group
 $security_group_id"
            return 0
        else
            echo "Found $eni_count network interface(s) still using security
 group $security_group_id"
            echo "Waiting 10 seconds before next check..."
            sleep 10
            ((attempt++))
        fi
    done

    echo "WARNING: Network interfaces may still be attached after $max_attempts
 attempts"
    echo "This is normal and the security group deletion will be retried"
    return 1
}

# Function to retry security group deletion with exponential backoff
retry_security_group_deletion() {
    local security_group_id="$1"
    local max_attempts=10
    local attempt=1
    local wait_time=5

    # Validate security group ID format
```

```
    if [[ ! "$security_group_id" =~ ^sg-[a-z0-9]{8,17}$ ]]; then
        echo "ERROR: Invalid security group ID format: $security_group_id"
        return 1
    fi

    while [[ $attempt -le $max_attempts ]]; do
        echo "Attempt $attempt/$max_attempts: Trying to delete security group
 $security_group_id"

        if execute_command "aws ec2 delete-security-group --group-id
 '$security_group_id'" "Delete security group (attempt $attempt)"; then
            echo "Successfully deleted security group $security_group_id"
            return 0
        else
            if [[ $attempt -eq $max_attempts ]]; then
                echo "FAILED: Could not delete security group $security_group_id
 after $max_attempts attempts"
                echo "This may be due to network interfaces that are still being
 cleaned up by AWS"
                echo "You can manually delete it later using: aws ec2 delete-
security-group --group-id $security_group_id"
                return 1
            else
                echo "Waiting $wait_time seconds before retry..."
                sleep $wait_time
                wait_time=$((wait_time * 2))  # Exponential backoff
                ((attempt++))
            fi
        fi
    done
}

# Function to cleanup resources with proper dependency handling
cleanup_resources() {
    echo ""
    echo "============================================="
    echo "CLEANUP PROCESS"
    echo "============================================="
    echo "The following resources were created:"
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "  - $resource"
    done
    echo ""
    echo "Auto-confirming cleanup of all created resources..."
```

```
   CLEANUP_CHOICE="y"

   if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
       echo "Starting cleanup process..."

       # Step 1: Scale service to 0 tasks first, then delete service
       if [[ " ${CREATED_RESOURCES[*]} " =~ " ECS Service: $SERVICE_NAME " ]];
then
           echo ""
           echo "Step 1: Scaling service to 0 tasks..."
           if execute_command "aws ecs update-service --cluster '$CLUSTER_NAME'
--service '$SERVICE_NAME' --desired-count 0" "Scale service to 0 tasks"; then
               echo "Waiting for service to stabilize after scaling to 0..."
               execute_command "aws ecs wait services-stable --cluster
'$CLUSTER_NAME' --services '$SERVICE_NAME'" "Wait for service to stabilize"

               echo "Deleting service..."
               execute_command "aws ecs delete-service --cluster '$CLUSTER_NAME'
--service '$SERVICE_NAME'" "Delete ECS service"
           else
               echo "WARNING: Failed to scale service. Attempting to delete
anyway..."
               execute_command "aws ecs delete-service --cluster '$CLUSTER_NAME'
--service '$SERVICE_NAME' --force" "Force delete ECS service"
           fi
       fi

       # Step 2: Wait a bit for tasks to fully terminate
       echo ""
       echo "Step 2: Waiting for tasks to fully terminate..."
       sleep 15

       # Step 3: Delete cluster
       if [[ " ${CREATED_RESOURCES[*]} " =~ " ECS Cluster: $CLUSTER_NAME " ]];
then
           echo ""
           echo "Step 3: Deleting cluster..."
           execute_command "aws ecs delete-cluster --cluster '$CLUSTER_NAME'"
"Delete ECS cluster"
       fi

       # Step 4: Wait for network interfaces to be cleaned up, then delete
security group
```

```
        if [[ -n "$SECURITY_GROUP_ID" && "$SECURITY_GROUP_ID" != "None" ]]; then
            echo ""
            echo "Step 4: Cleaning up security group..."

            # First, wait for network interfaces to be cleaned up
            wait_for_network_interfaces_cleanup "$SECURITY_GROUP_ID"

            # Then retry security group deletion with backoff
            retry_security_group_deletion "$SECURITY_GROUP_ID"
        fi

        # Step 5: Clean up task definition (deregister all revisions)
        if [[ " ${CREATED_RESOURCES[*]} " =~ " Task Definition: $TASK_FAMILY
 " ]]; then
            echo ""
            echo "Step 5: Deregistering task definition revisions..."

            # Get all revisions of the task definition
            local revisions
            revisions=$(aws ecs list-task-definitions --family-prefix
 "$TASK_FAMILY" --query "taskDefinitionArns" --output text 2>/dev/null || echo
 "")

            if [[ -n "$revisions" && "$revisions" != "None" ]]; then
                for revision_arn in $revisions; do
                    echo "Deregistering task definition: $revision_arn"
                    execute_command "aws ecs deregister-task-definition --task-
definition '$revision_arn'" "Deregister task definition $revision_arn" || true
                done
            else
                echo "No task definition revisions found to deregister"
            fi
        fi

        echo ""
        echo "============================================"
        echo "CLEANUP COMPLETED"
        echo "============================================"
        echo "All resources have been cleaned up successfully!"

    else
        echo "Cleanup skipped. Resources remain active."
        echo ""
        echo "To clean up manually later, use the following commands in order:"
```

```
        echo "1. Scale service to 0: aws ecs update-service --cluster
 $CLUSTER_NAME --service $SERVICE_NAME --desired-count 0"
        echo "2. Wait for stability: aws ecs wait services-stable --cluster
 $CLUSTER_NAME --services $SERVICE_NAME"
        echo "3. Delete service: aws ecs delete-service --cluster $CLUSTER_NAME
 --service $SERVICE_NAME"
        echo "4. Delete cluster: aws ecs delete-cluster --cluster $CLUSTER_NAME"
        echo "5. Wait 2-3 minutes, then delete security group: aws ec2 delete-
security-group --group-id $SECURITY_GROUP_ID"
        if [[ " ${CREATED_RESOURCES[*]} " =~ " Task Definition: $TASK_FAMILY
 " ]]; then
            echo "6. Deregister task definitions: aws ecs list-task-definitions
 --family-prefix $TASK_FAMILY"
            echo "   Then for each ARN: aws ecs deregister-task-definition --
task-definition <ARN>"
        fi
    fi
}

# Trap to handle script interruption
trap cleanup_resources EXIT

echo "Using random identifier: $RANDOM_ID"
echo "Cluster name: $CLUSTER_NAME"
echo "Service name: $SERVICE_NAME"
echo "Task family: $TASK_FAMILY"

# Step 1: Ensure ECS task execution role exists
echo ""
echo "==========================================="
echo "STEP 1: VERIFY ECS TASK EXECUTION ROLE"
echo "==========================================="

ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
if [[ ! "$ACCOUNT_ID" =~ ^[0-9]{12}$ ]]; then
    echo "ERROR: Invalid AWS Account ID retrieved: $ACCOUNT_ID"
    exit 1
fi

EXECUTION_ROLE_ARN="arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskExecutionRole"

# Check if role exists
if aws iam get-role --role-name ecsTaskExecutionRole >/dev/null 2>&1; then
    echo "ECS task execution role already exists"
```

```
else
    echo "Creating ECS task execution role..."

    # Create trust policy with strict validation
    cat > trust-policy.json << 'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF

    # Validate JSON before using
    if ! jq empty trust-policy.json 2>/dev/null; then
        echo "ERROR: Invalid JSON in trust policy"
        rm -f trust-policy.json
        exit 1
    fi

    execute_command "aws iam create-role --role-name ecsTaskExecutionRole --
assume-role-policy-document file://trust-policy.json" "Create ECS task execution
 role"

    aws iam tag-role --role-name ecsTaskExecutionRole --tags
 Key=project,Value=doc-smith Key=tutorial,Value=amazon-ecs-fargate-linux

    execute_command "aws iam attach-role-policy --role-name
 ecsTaskExecutionRole --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy" "Attach ECS task execution policy"

    # Clean up temporary file securely
    shred -vfz -n 3 trust-policy.json 2>/dev/null || rm -f trust-policy.json

    CREATED_RESOURCES+=("IAM Role: ecsTaskExecutionRole")
fi

# Step 2: Create ECS cluster
```

```
echo ""
echo "============================================="
echo "STEP 2: CREATE ECS CLUSTER"
echo "============================================="

CLUSTER_OUTPUT=$(execute_command "aws ecs create-cluster --cluster-name
 '$CLUSTER_NAME' --tags key=project,value=doc-smith key=tutorial,value=amazon-
ecs-fargate-linux" "Create ECS cluster")
check_for_aws_errors "$CLUSTER_OUTPUT" "Create ECS cluster"

CREATED_RESOURCES+=("ECS Cluster: $CLUSTER_NAME")

# Step 3: Create task definition
echo ""
echo "============================================="
echo "STEP 3: CREATE TASK DEFINITION"
echo "============================================="

# Create task definition JSON with validated inputs
cat > task-definition.json << EOF
{
    "family": "$TASK_FAMILY",
    "networkMode": "awsvpc",
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "256",
    "memory": "512",
    "executionRoleArn": "$EXECUTION_ROLE_ARN",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "public.ecr.aws/docker/library/httpd:2.4-alpine",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": ["sh", "-c"],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
```

```
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container
 in Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html
 && httpd-foreground\""
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/fargate-sample",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ]
}
EOF

# Validate JSON before using
if ! jq empty task-definition.json 2>/dev/null; then
    echo "ERROR: Invalid JSON in task definition"
    rm -f task-definition.json
    exit 1
fi

TASK_DEF_OUTPUT=$(execute_command "aws ecs register-task-definition --cli-input-
json file://task-definition.json" "Register task definition")
check_for_aws_errors "$TASK_DEF_OUTPUT" "Register task definition"

# Clean up temporary file securely
shred -vfz -n 3 task-definition.json 2>/dev/null || rm -f task-definition.json

CREATED_RESOURCES+=("Task Definition: $TASK_FAMILY")

# Step 4: Set up networking
echo ""
echo "==========================================="
echo "STEP 4: SET UP NETWORKING"
echo "==========================================="

# Get default VPC ID
VPC_ID=$(aws ec2 describe-vpcs --filters "Name=is-default,Values=true" --query
 "Vpcs[0].VpcId" --output text)
if [[ "$VPC_ID" == "None" || -z "$VPC_ID" ]]; then
```

```
    echo "ERROR: No default VPC found. Please create a default VPC or specify a
 custom VPC."
    exit 1
fi

# Validate VPC ID format
if [[ ! "$VPC_ID" =~ ^vpc-[a-z0-9]{8,17}$ ]]; then
    echo "ERROR: Invalid VPC ID format: $VPC_ID"
    exit 1
fi

echo "Using default VPC: $VPC_ID"

# Create security group with restricted access
# Note: This allows HTTP access from anywhere for demo purposes
# In production, restrict source to specific IP ranges or security groups
SECURITY_GROUP_OUTPUT=$(execute_command "aws ec2 create-security-group --
group-name '$SECURITY_GROUP_NAME' --description 'Security group for ECS
 Fargate tutorial - HTTP access' --vpc-id '$VPC_ID' --tag-specifications
 'ResourceType=security-group,Tags=[{Key=project,Value=doc-smith},
{Key=tutorial,Value=amazon-ecs-fargate-linux}]'" "Create security group")
check_for_aws_errors "$SECURITY_GROUP_OUTPUT" "Create security group"

SECURITY_GROUP_ID=$(echo "$SECURITY_GROUP_OUTPUT" | grep -o '"GroupId": "[^"]*"'
 | head -1 | cut -d'"' -f4)
if [[ -z "$SECURITY_GROUP_ID" || "$SECURITY_GROUP_ID" == "None" ]]; then
    SECURITY_GROUP_ID=$(aws ec2 describe-security-groups --group-names
 "$SECURITY_GROUP_NAME" --filters "Name=vpc-id,Values=$VPC_ID" --query
 "SecurityGroups[0].GroupId" --output text)
fi

# Validate security group ID format
if [[ ! "$SECURITY_GROUP_ID" =~ ^sg-[a-z0-9]{8,17}$ ]]; then
    echo "ERROR: Invalid security group ID format: $SECURITY_GROUP_ID"
    exit 1
fi

echo "Created security group: $SECURITY_GROUP_ID"
CREATED_RESOURCES+=("Security Group: $SECURITY_GROUP_ID")

# Add HTTP inbound rule
# WARNING: This allows HTTP access from anywhere (0.0.0.0/0)
# In production environments, restrict this to specific IP ranges
```

```
execute_command "aws ec2 authorize-security-group-ingress --group-id
 '$SECURITY_GROUP_ID' --protocol tcp --port 80 --cidr 0.0.0.0/0" "Add HTTP
 inbound rule to security group"

# Get subnet IDs from default VPC
echo "Getting subnet IDs from default VPC..."
SUBNET_IDS_RAW=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=$VPC_ID"
 --query "Subnets[*].SubnetId" --output text)
if [[ -z "$SUBNET_IDS_RAW" ]]; then
    echo "ERROR: No subnets found in default VPC"
    exit 1
fi

# Convert to proper comma-separated format, handling both spaces and tabs
SUBNET_IDS_COMMA=$(echo "$SUBNET_IDS_RAW" | tr -s '[:space:]' ',' | sed 's/,$//')
echo "Raw subnet IDs: $SUBNET_IDS_RAW"
echo "Formatted subnet IDs: $SUBNET_IDS_COMMA"

# Validate subnet IDs format
if [[ ! "$SUBNET_IDS_COMMA" =~ ^subnet-[a-z0-9]+(,subnet-[a-z0-9]+)*$ ]]; then
    echo "ERROR: Invalid subnet ID format: $SUBNET_IDS_COMMA"
    exit 1
fi

# Step 5: Create ECS service
echo ""
echo "==========================================="
echo "STEP 5: CREATE ECS SERVICE"
echo "==========================================="

# Create the service with proper JSON formatting for network configuration
SERVICE_CMD="aws ecs create-service --cluster '$CLUSTER_NAME' --service-name
 '$SERVICE_NAME' --task-definition '$TASK_FAMILY' --desired-count 1 --launch-
type FARGATE --network-configuration '{\"awsvpcConfiguration\":{\"subnets
\":[\"$(echo "$SUBNET_IDS_COMMA" | sed 's/,/","/g')\"],\"securityGroups
\":[\"$SECURITY_GROUP_ID\"],\"assignPublicIp\":\"ENABLED\"}}' --tags
 key=project,value=doc-smith key=tutorial,value=amazon-ecs-fargate-linux"

echo "Service creation command: $SERVICE_CMD"

SERVICE_OUTPUT=$(execute_command "$SERVICE_CMD" "Create ECS service")
check_for_aws_errors "$SERVICE_OUTPUT" "Create ECS service"

CREATED_RESOURCES+=("ECS Service: $SERVICE_NAME")
```

```
# Step 6: Wait for service to stabilize and get public IP
echo ""
echo "============================================="
echo "STEP 6: WAIT FOR SERVICE AND GET PUBLIC IP"
echo "============================================="

echo "Waiting for service to stabilize (this may take a few minutes)..."
execute_command "aws ecs wait services-stable --cluster '$CLUSTER_NAME' --
services '$SERVICE_NAME'" "Wait for service to stabilize"

# Get task ARN
TASK_ARN=$(aws ecs list-tasks --cluster "$CLUSTER_NAME" --service-name
 "$SERVICE_NAME" --query "taskArns[0]" --output text)
if [[ "$TASK_ARN" == "None" || -z "$TASK_ARN" ]]; then
    echo "ERROR: No running tasks found for service"
    exit 1
fi

echo "Task ARN: $TASK_ARN"

# Get network interface ID
ENI_ID=$(aws ecs describe-tasks --cluster "$CLUSTER_NAME" --tasks "$TASK_ARN"
 --query "tasks[0].attachments[0].details[?name=='networkInterfaceId'].value" --
output text)
if [[ "$ENI_ID" == "None" || -z "$ENI_ID" ]]; then
    echo "ERROR: Could not retrieve network interface ID"
    exit 1
fi

# Validate ENI ID format
if [[ ! "$ENI_ID" =~ ^eni-[a-z0-9]{8,17}$ ]]; then
    echo "ERROR: Invalid network interface ID format: $ENI_ID"
    exit 1
fi

echo "Network Interface ID: $ENI_ID"

# Get public IP
PUBLIC_IP=$(aws ec2 describe-network-interfaces --network-interface-ids "$ENI_ID"
 --query "NetworkInterfaces[0].Association.PublicIp" --output text)
if [[ "$PUBLIC_IP" == "None" || -z "$PUBLIC_IP" ]]; then
    echo "WARNING: No public IP assigned to the task"
    echo "The task may be in a private subnet or public IP assignment failed"
```

```
else
    # Validate IP format
    if [[ ! "$PUBLIC_IP" =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]];
 then
        echo "ERROR: Invalid IP address format: $PUBLIC_IP"
        PUBLIC_IP=""
    else
        echo ""
        echo "========================================="
        echo "SUCCESS! APPLICATION IS RUNNING"
        echo "========================================="
        echo "Your application is available at: http://$PUBLIC_IP"
        echo "You can test it by opening this URL in your browser"
        echo ""
    fi
fi

# Display service information
echo ""
echo "========================================="
echo "SERVICE INFORMATION"
echo "========================================="
execute_command "aws ecs describe-services --cluster '$CLUSTER_NAME' --services
 '$SERVICE_NAME'" "Get service details"

echo ""
echo "========================================="
echo "TUTORIAL COMPLETED SUCCESSFULLY"
echo "========================================="
echo "Resources created:"
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "  - $resource"
done

if [[ -n "$PUBLIC_IP" && "$PUBLIC_IP" != "None" ]]; then
    echo ""
    echo "Application URL: http://$PUBLIC_IP"
fi

echo ""
echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.

- AttachRolePolicy
- AuthorizeSecurityGroupIngress
- CreateCluster
- CreateRole
- CreateSecurityGroup
- CreateService
- DeleteCluster
- DeleteSecurityGroup
- DeleteService
- DeregisterTaskDefinition
- DescribeNetworkInterfaces
- DescribeSecurityGroups
- DescribeServices
- DescribeSubnets
- DescribeTasks
- DescribeVpcs
- GetCallerIdentity
- GetRole
- ListTaskDefinitions
- ListTasks
- RegisterTaskDefinition
- UpdateService
- Wait

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a rest API with function proxy integration

The following code example shows how to:

- Create an IAM role for Lambda execution

- Create and deploy a Lambda function
- Create a REST API
- Configure Lambda proxy integration
- Deploy and test the API
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

set -euo pipefail

# Simple API Gateway Lambda Integration Script
# This script creates a REST API with Lambda proxy integration

# Generate random identifiers
FUNCTION_NAME="GetStartedLambdaProxyIntegration-$(openssl rand -hex 4)"
ROLE_NAME="GetStartedLambdaBasicExecutionRole-$(openssl rand -hex 4)"
API_NAME="LambdaProxyAPI-$(openssl rand -hex 4)"

# Get AWS account info
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REGION=$(aws configure get region || echo "us-east-1")

# Validate inputs
if [[ -z "$ACCOUNT_ID" ]] || [[ -z "$REGION" ]]; then
    echo "Error: Failed to retrieve AWS account information" >&2
    exit 1
fi

echo "Creating Lambda function code..."
```

```
# Create Lambda function code with input validation
cat > lambda_function.py << 'EOF'
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        logger.info("Received event: %s", json.dumps(event))

        greeter = 'World'

        # Safely retrieve greeter from query string parameters
        query_params = event.get('queryStringParameters') or {}
        if isinstance(query_params, dict) and 'greeter' in query_params:
            greeter_value = query_params.get('greeter')
            if isinstance(greeter_value, str) and greeter_value:
                greeter = greeter_value

        # Safely retrieve greeter from multi-value headers
        multi_headers = event.get('multiValueHeaders') or {}
        if isinstance(multi_headers, dict) and 'greeter' in multi_headers:
            greeter_list = multi_headers.get('greeter', [])
            if isinstance(greeter_list, list) and greeter_list:
                greeter = " and ".join(str(g) for g in greeter_list if g)

        # Safely retrieve greeter from headers
        headers = event.get('headers') or {}
        if isinstance(headers, dict) and 'greeter' in headers:
            greeter_value = headers.get('greeter')
            if isinstance(greeter_value, str) and greeter_value:
                greeter = greeter_value

        # Safely retrieve greeter from body
        body = event.get('body')
        if body and isinstance(body, str):
            try:
                body_dict = json.loads(body)
                if isinstance(body_dict, dict) and 'greeter' in body_dict:
                    greeter_value = body_dict.get('greeter')
                    if isinstance(greeter_value, str) and greeter_value:
                        greeter = greeter_value
```

```
            except (json.JSONDecodeError, ValueError) as e:
                logger.warning("Failed to parse body: %s", str(e))

        # Sanitize greeter to prevent injection
        greeter = greeter.replace('"', '\\"').replace("'", "\\'")

        response = {
            "statusCode": 200,
            "headers": {
                "Content-Type": "application/json"
            },
            "body": json.dumps({"message": f"Hello, {greeter}!"})
        }

        logger.info("Response: %s", json.dumps(response))
        return response

    except Exception as e:
        logger.error("Unexpected error: %s", str(e), exc_info=True)
        return {
            "statusCode": 500,
            "headers": {
                "Content-Type": "application/json"
            },
            "body": json.dumps({"error": "Internal server error"})
        }
EOF

# Create deployment package
zip -q function.zip lambda_function.py || {
    echo "Error: Failed to create function.zip" >&2
    exit 1
}

echo "Creating IAM role..."

# Create IAM trust policy
cat > trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role with error handling
aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document file://trust-policy.json \
    --description "Temporary role for Lambda execution" || {
    echo "Error: Failed to create IAM role" >&2
    exit 1
}

aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=apigateway-lambda-integration

# Attach execution policy
aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || {
    echo "Error: Failed to attach IAM policy" >&2
    exit 1
}

# Wait for role propagation
sleep 15

echo "Creating Lambda function..."

# Create Lambda function with Python 3.11 (more recent runtime)
aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime python3.11 \
    --role "arn:aws:iam::$ACCOUNT_ID:role/$ROLE_NAME" \
    --handler lambda_function.lambda_handler \
    --zip-file fileb://function.zip \
    --timeout 30 \
    --memory-size 128 \
    --environment "Variables={LOG_LEVEL=INFO}" \
```

```
    --tags project=doc-smith,tutorial=apigateway-lambda-integration || {
    echo "Error: Failed to create Lambda function" >&2
    exit 1
}

echo "Creating API Gateway..."

# Create REST API with minimum logging
API_RESPONSE=$(aws apigateway create-rest-api \
    --name "$API_NAME" \
    --endpoint-configuration types=REGIONAL \
    --description "API for Lambda proxy integration tutorial" \
    --tags project=doc-smith,tutorial=apigateway-lambda-integration \
    --output json)

API_ID=$(echo "$API_RESPONSE" | grep -o '"id": "[^"]*"' | head -1 | cut -d'"' -
f4)

if [[ -z "$API_ID" ]]; then
    echo "Error: Failed to create API Gateway" >&2
    exit 1
fi

# Get root resource ID
ROOT_RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
 'items[?path==`/`].id' --output text)

# Create helloworld resource
aws apigateway create-resource \
    --rest-api-id "$API_ID" \
    --parent-id "$ROOT_RESOURCE_ID" \
    --path-part helloworld || {
    echo "Error: Failed to create resource" >&2
    exit 1
}

# Get resource ID
RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
 "items[?pathPart=='helloworld'].id" --output text)

# Create ANY method with no authorization (intentional for tutorial)
aws apigateway put-method \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
```

```
    --http-method ANY \
    --authorization-type NONE || {
    echo "Error: Failed to create method" >&2
    exit 1
}

# Set up Lambda proxy integration
LAMBDA_URI="arn:aws:apigateway:$REGION:lambda:path/2015-03-31/functions/
arn:aws:lambda:$REGION:$ACCOUNT_ID:function:$FUNCTION_NAME/invocations"

aws apigateway put-integration \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
    --http-method ANY \
    --type AWS_PROXY \
    --integration-http-method POST \
    --uri "$LAMBDA_URI" || {
    echo "Error: Failed to create integration" >&2
    exit 1
}

# Grant API Gateway permission to invoke Lambda
STATEMENT_ID="apigateway-invoke-$(openssl rand -hex 4)"
SOURCE_ARN="arn:aws:execute-api:$REGION:$ACCOUNT_ID:$API_ID/*/*"

aws lambda add-permission \
    --function-name "$FUNCTION_NAME" \
    --statement-id "$STATEMENT_ID" \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "$SOURCE_ARN" || {
    echo "Error: Failed to add Lambda permission" >&2
    exit 1
}

# Deploy API
aws apigateway create-deployment \
    --rest-api-id "$API_ID" \
    --stage-name test \
    --description "Test deployment" || {
    echo "Error: Failed to deploy API" >&2
    exit 1
}
```

```
echo "Testing API..."

# Test the API
INVOKE_URL="https://$API_ID.execute-api.$REGION.amazonaws.com/test/helloworld"

echo "API URL: $INVOKE_URL"

# Test with query parameter (with proper URL encoding)
echo "Testing with query parameter:"
curl -s -X GET "$INVOKE_URL?greeter=John" | jq . 2>/dev/null || curl -s -X GET
 "$INVOKE_URL?greeter=John"
echo ""

# Test with header
echo "Testing with header:"
curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -H 'greeter: John' | jq . 2>/dev/null || curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -H 'greeter: John'
echo ""

# Test with body
echo "Testing with POST body:"
curl -s -X POST "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}' | jq . 2>/dev/null || curl -s -X POST "$INVOKE_URL"
 \
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}'
echo ""

echo "Tutorial completed! API is available at: $INVOKE_URL"

# Cleanup
echo "Cleaning up resources..."

# Delete API
aws apigateway delete-rest-api --rest-api-id "$API_ID" || echo "Warning: Failed
 to delete API" >&2

# Delete Lambda function
aws lambda delete-function --function-name "$FUNCTION_NAME" || echo "Warning:
 Failed to delete Lambda function" >&2
```

```
# Detach policy and delete role
aws iam detach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || echo "Warning: Failed to detach policy" >&2

aws iam delete-role --role-name "$ROLE_NAME" || echo "Warning: Failed to delete
 role" >&2

# Clean up local files securely
rm -f lambda_function.py function.zip trust-policy.json

echo "Cleanup completed!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddPermission
  - AttachRolePolicy
  - CreateDeployment
  - CreateFunction
  - CreateResource
  - CreateRestApi
  - CreateRole
  - DeleteFunction
  - DeleteRestApi
  - DeleteRole
  - DetachRolePolicy
  - GetCallerIdentity
  - GetResources
  - GetRestApis
  - PutIntegration
  - PutMethod

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create read-only and read-write IAM users using an AWS SDK

The following code example shows how to create users and attach policies to them.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create two IAM users.
- Attach a policy for one user to get and put objects in an Amazon S3 bucket.
- Attach a policy for the second user to get objects from the bucket.
- Get different permissions to the bucket based on user credentials.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM user actions.

```
import logging
import time

import boto3
from botocore.exceptions import ClientError

import access_key_wrapper
```

```
import policy_wrapper

logger = logging.getLogger(__name__)
iam = boto3.resource("iam")

def create_user(user_name):
    """
    Creates a user. By default, a user has no permissions or access keys.

    :param user_name: The name of the user.
    :return: The newly created user.
    """
    try:
        user = iam.create_user(UserName=user_name)
        logger.info("Created user %s.", user.name)
    except ClientError:
        logger.exception("Couldn't create user %s.", user_name)
        raise
    else:
        return user



def update_user(user_name, new_user_name):
    """
    Updates a user's name.

    :param user_name: The current name of the user to update.
    :param new_user_name: The new name to assign to the user.
    :return: The updated user.
    """
    try:
        user = iam.User(user_name)
        user.update(NewUserName=new_user_name)
        logger.info("Renamed %s to %s.", user_name, new_user_name)
    except ClientError:
        logger.exception("Couldn't update name for user %s.", user_name)
        raise
    return user



def list_users():
    """
```

```
    Lists the users in the current account.

    :return: The list of users.
    """
    try:
        users = list(iam.users.all())
        logger.info("Got %s users.", len(users))
    except ClientError:
        logger.exception("Couldn't get users.")
        raise
    else:
        return users



def delete_user(user_name):
    """
    Deletes a user. Before a user can be deleted, all associated resources,
    such as access keys and policies, must be deleted or detached.

    :param user_name: The name of the user.
    """
    try:
        iam.User(user_name).delete()
        logger.info("Deleted user %s.", user_name)
    except ClientError:
        logger.exception("Couldn't delete user %s.", user_name)
        raise



def attach_policy(user_name, policy_arn):
    """
    Attaches a policy to a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to user %s.", policy_arn,
 user_name)
```

```
        raise



def detach_policy(user_name, policy_arn):
    """
    Detaches a policy from a user.

    :param user_name: The name of the user.
    :param policy_arn: The Amazon Resource Name (ARN) of the policy.
    """
    try:
        iam.User(user_name).detach_policy(PolicyArn=policy_arn)
        logger.info("Detached policy %s from user %s.", policy_arn, user_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from user %s.", policy_arn, user_name
        )
        raise
```

Create functions that wrap IAM policy actions.

```
import json
import logging
import operator
import pprint
import time

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource("iam")

def create_policy(name, description, actions, resource_arn):
    """
    Creates a policy that contains a single statement.

    :param name: The name of the policy to create.
    :param description: The description of the policy.
```

```
    :param actions: The actions allowed by the policy. These typically take the
                    form of service:action, such as s3:PutObject.
    :param resource_arn: The Amazon Resource Name (ARN) of the resource this
 policy
                         applies to. This ARN can contain wildcards, such as
                         'arn:aws:s3:::amzn-s3-demo-bucket/*' to allow actions on
 all objects
                         in the bucket named 'amzn-s3-demo-bucket'.
    :return: The newly created policy.
    """
    policy_doc = {
        "Version":"2012-10-17",
        "Statement": [{"Effect": "Allow", "Action": actions, "Resource":
 resource_arn}],
    }
    try:
        policy = iam.create_policy(
            PolicyName=name,
            Description=description,
            PolicyDocument=json.dumps(policy_doc),
        )
        logger.info("Created policy %s.", policy.arn)
    except ClientError:
        logger.exception("Couldn't create policy %s.", name)
        raise
    else:
        return policy



def delete_policy(policy_arn):
    """
    Deletes a policy.

    :param policy_arn: The ARN of the policy to delete.
    """
    try:
        iam.Policy(policy_arn).delete()
        logger.info("Deleted policy %s.", policy_arn)
    except ClientError:
        logger.exception("Couldn't delete policy %s.", policy_arn)
        raise
```

Create functions that wrap IAM access key actions.

```
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

iam = boto3.resource("iam")

def create_key(user_name):
    """
    Creates an access key for the specified user. Each user can have a
    maximum of two keys.

    :param user_name: The name of the user.
    :return: The created access key.
    """
    try:
        key_pair = iam.User(user_name).create_access_key_pair()
        logger.info(
            "Created access key pair for %s. Key ID is %s.",
            key_pair.user_name,
            key_pair.id,
        )
    except ClientError:
        logger.exception("Couldn't create access key pair for %s.", user_name)
        raise
    else:
        return key_pair



def delete_key(user_name, key_id):
    """
    Deletes a user's access key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to delete.
    """
```

```
    try:
        key = iam.AccessKey(user_name, key_id)
        key.delete()
        logger.info("Deleted access key %s for %s.", key.id, key.user_name)
    except ClientError:
        logger.exception("Couldn't delete key %s for %s", key_id, user_name)
        raise
```

Use the wrapper functions to create users with differing policies and use their credentials to access an Amazon S3 bucket.

```
def usage_demo():
    """
    Shows how to manage users, keys, and policies.
    This demonstration creates two users: one user who can put and get objects in
 an
    Amazon S3 bucket, and another user who can only get objects from the bucket.
    The demo then shows how the users can perform only the actions they are
 permitted
    to perform.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    print("-" * 88)
    print("Welcome to the AWS Identity and Account Management user demo.")
    print("-" * 88)
    print(
        "Users can have policies and roles attached to grant them specific "
        "permissions."
    )
    s3 = boto3.resource("s3")
    bucket = s3.create_bucket(
        Bucket=f"demo-iam-bucket-{time.time_ns()}",
        CreateBucketConfiguration={
            "LocationConstraint": s3.meta.client.meta.region_name
        },
    )
    print(f"Created an Amazon S3 bucket named {bucket.name}.")
    user_read_writer = create_user("demo-iam-read-writer")
    user_reader = create_user("demo-iam-reader")
```

```
    print(f"Created two IAM users: {user_read_writer.name} and
{user_reader.name}")
    update_user(user_read_writer.name, "demo-iam-creator")
    update_user(user_reader.name, "demo-iam-getter")
    users = list_users()
    user_read_writer = next(
        user for user in users if user.user_id == user_read_writer.user_id
    )
    user_reader = next(user for user in users if user.user_id ==
user_reader.user_id)
    print(
        f"Changed the names of the users to {user_read_writer.name} "
        f"and {user_reader.name}."
    )

    read_write_policy = policy_wrapper.create_policy(
        "demo-iam-read-write-policy",
        "Grants rights to create and get an object in the demo bucket.",
        ["s3:PutObject", "s3:GetObject"],
        f"arn:aws:s3:::{bucket.name}/*",
    )
    print(
        f"Created policy {read_write_policy.policy_name} with ARN:
{read_write_policy.arn}"
    )
    print(read_write_policy.description)
    read_policy = policy_wrapper.create_policy(
        "demo-iam-read-policy",
        "Grants rights to get an object from the demo bucket.",
        "s3:GetObject",
        f"arn:aws:s3:::{bucket.name}/*",
    )
    print(f"Created policy {read_policy.policy_name} with ARN:
{read_policy.arn}")
    print(read_policy.description)
    attach_policy(user_read_writer.name, read_write_policy.arn)
    print(f"Attached {read_write_policy.policy_name} to
{user_read_writer.name}.")
    attach_policy(user_reader.name, read_policy.arn)
    print(f"Attached {read_policy.policy_name} to {user_reader.name}.")

    user_read_writer_key = access_key_wrapper.create_key(user_read_writer.name)
    print(f"Created access key pair for {user_read_writer.name}.")
    user_reader_key = access_key_wrapper.create_key(user_reader.name)
```

```
    print(f"Created access key pair for {user_reader.name}.")

    s3_read_writer_resource = boto3.resource(
        "s3",
        aws_access_key_id=user_read_writer_key.id,
        aws_secret_access_key=user_read_writer_key.secret,
    )
    demo_object_key = f"object-{time.time_ns()}"
    demo_object = None
    while demo_object is None:
        try:
            demo_object = s3_read_writer_resource.Bucket(bucket.name).put_object(
                Key=demo_object_key, Body=b"AWS IAM demo object content!"
            )
        except ClientError as error:
            if error.response["Error"]["Code"] == "InvalidAccessKeyId":
                print("Access key not yet available. Waiting...")
                time.sleep(1)
            else:
                raise
    print(
        f"Put {demo_object_key} into {bucket.name} using "
        f"{user_read_writer.name}'s credentials."
    )

    read_writer_object = s3_read_writer_resource.Bucket(bucket.name).Object(
        demo_object_key
    )
    read_writer_content = read_writer_object.get()["Body"].read()
    print(f"Got object {read_writer_object.key} using read-writer user's
credentials.")
    print(f"Object content: {read_writer_content}")

    s3_reader_resource = boto3.resource(
        "s3",
        aws_access_key_id=user_reader_key.id,
        aws_secret_access_key=user_reader_key.secret,
    )
    demo_content = None
    while demo_content is None:
        try:
            demo_object =
s3_reader_resource.Bucket(bucket.name).Object(demo_object_key)
            demo_content = demo_object.get()["Body"].read()
```

```
            print(f"Got object {demo_object.key} using reader user's
credentials.")
            print(f"Object content: {demo_content}")
        except ClientError as error:
            if error.response["Error"]["Code"] == "InvalidAccessKeyId":
                print("Access key not yet available. Waiting...")
                time.sleep(1)
            else:
                raise

    try:
        demo_object.delete()
    except ClientError as error:
        if error.response["Error"]["Code"] == "AccessDenied":
            print("-" * 88)
            print(
                "Tried to delete the object using the reader user's credentials.
"
                "Got expected AccessDenied error because the reader is not "
                "allowed to delete objects."
            )
            print("-" * 88)

    access_key_wrapper.delete_key(user_reader.name, user_reader_key.id)
    detach_policy(user_reader.name, read_policy.arn)
    policy_wrapper.delete_policy(read_policy.arn)
    delete_user(user_reader.name)
    print(f"Deleted keys, detached and deleted policy, and deleted
{user_reader.name}.")

    access_key_wrapper.delete_key(user_read_writer.name, user_read_writer_key.id)
    detach_policy(user_read_writer.name, read_write_policy.arn)
    policy_wrapper.delete_policy(read_write_policy.arn)
    delete_user(user_read_writer.name)
    print(
        f"Deleted keys, detached and deleted policy, and deleted
{user_read_writer.name}."
    )

    bucket.objects.delete()
    bucket.delete()
    print(f"Emptied and deleted {bucket.name}.")
    print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - AttachUserPolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteUser
  - DetachUserPolicy
  - ListUsers
  - UpdateUser

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Creating a container service for virtual machine instances

The following code example shows how to:

- Create an ECS cluster
- Create and monitor a service
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# ECS EC2 Launch Type Tutorial Script - UPDATED VERSION
# This script demonstrates creating an ECS cluster, launching a container
 instance,
# registering a task definition, and creating a service using the EC2 launch
 type.
# Updated to match the tutorial draft with nginx web server and service creation.
#
# - UPDATED: Changed from sleep task to nginx web server with service

set -e  # Exit on any error

# Configuration
SCRIPT_NAME="ecs-ec2-tutorial"
LOG_FILE="${SCRIPT_NAME}-$(date +%Y%m%d-%H%M%S).log"
CLUSTER_NAME="tutorial-cluster-$(openssl rand -hex 4)"
TASK_FAMILY="nginx-task-$(openssl rand -hex 4)"
SERVICE_NAME="nginx-service-$(openssl rand -hex 4)"
KEY_PAIR_NAME="ecs-tutorial-key-$(openssl rand -hex 4)"
SECURITY_GROUP_NAME="ecs-tutorial-sg-$(openssl rand -hex 4)"

# Tags
PROJECT_TAG="doc-smith"
TUTORIAL_TAG="ecs-ec2"

# Get current AWS region dynamically
AWS_REGION=$(aws configure get region || echo "us-east-1")

# Resource tracking arrays
CREATED_RESOURCES=()
CLEANUP_ORDER=()

# Logging function
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

# Error handling function
handle_error() {
    local exit_code=$?
    log "ERROR: Script failed with exit code $exit_code"
    log "ERROR: Last command: $BASH_COMMAND"
```

```
    echo ""
    echo "========================================="
    echo "ERROR OCCURRED - ATTEMPTING CLEANUP"
    echo "========================================="
    echo "Resources created before error:"
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "  - $resource"
    done

    cleanup_resources
    exit $exit_code
}

# Set error trap
trap handle_error ERR

# FIXED: Enhanced cleanup function with proper error handling and logging
cleanup_resources() {
    log "Starting cleanup process..."
    local cleanup_errors=0

    # Delete service first (this will stop tasks automatically)
    if [[ -n "${SERVICE_ARN:-}" ]]; then
        log "Updating service to desired count 0: $SERVICE_NAME"
        if ! aws ecs update-service --cluster "$CLUSTER_NAME" --service
 "$SERVICE_NAME" --desired-count 0 2>>"$LOG_FILE"; then
            log "WARNING: Failed to update service desired count to 0"
            ((cleanup_errors++))
        else
            log "Waiting for service tasks to stop..."
            sleep 30  # Give time for tasks to stop
        fi

        log "Deleting service: $SERVICE_NAME"
        if ! aws ecs delete-service --cluster "$CLUSTER_NAME" --service
 "$SERVICE_NAME" 2>>"$LOG_FILE"; then
            log "WARNING: Failed to delete service $SERVICE_NAME"
            ((cleanup_errors++))
        fi
    fi

    # Stop and delete any remaining tasks
    if [[ -n "${TASK_ARN:-}" ]]; then
```

```
        log "Stopping task: $TASK_ARN"
        if ! aws ecs stop-task --cluster "$CLUSTER_NAME" --task "$TASK_ARN" --
reason "Tutorial cleanup" 2>>"$LOG_FILE"; then
            log "WARNING: Failed to stop task $TASK_ARN"
            ((cleanup_errors++))
        else
            log "Waiting for task to stop..."
            if ! aws ecs wait tasks-stopped --cluster "$CLUSTER_NAME" --tasks
 "$TASK_ARN" 2>>"$LOG_FILE"; then
                log "WARNING: Task stop wait failed for $TASK_ARN"
                ((cleanup_errors++))
            fi
        fi
    fi

    # Deregister task definition
    if [[ -n "${TASK_DEFINITION_ARN:-}" ]]; then
        log "Deregistering task definition: $TASK_DEFINITION_ARN"
        if ! aws ecs deregister-task-definition --task-definition
 "$TASK_DEFINITION_ARN" 2>>"$LOG_FILE"; then
            log "WARNING: Failed to deregister task definition
 $TASK_DEFINITION_ARN"
            ((cleanup_errors++))
        fi
    fi

    # Terminate EC2 instance
    if [[ -n "${INSTANCE_ID:-}" ]]; then
        log "Terminating EC2 instance: $INSTANCE_ID"
        if ! aws ec2 terminate-instances --instance-ids "$INSTANCE_ID"
 2>>"$LOG_FILE"; then
            log "WARNING: Failed to terminate instance $INSTANCE_ID"
            ((cleanup_errors++))
        else
            log "Waiting for instance to terminate..."
            if ! aws ec2 wait instance-terminated --instance-ids "$INSTANCE_ID"
 2>>"$LOG_FILE"; then
                log "WARNING: Instance termination wait failed for $INSTANCE_ID"
                ((cleanup_errors++))
            fi
        fi
    fi

    # Delete security group with retry logic
```

```
   if [[ -n "${SECURITY_GROUP_ID:-}" ]]; then
       log "Deleting security group: $SECURITY_GROUP_ID"
       local retry_count=0
       local max_retries=3

       while [[ $retry_count -lt $max_retries ]]; do
           if aws ec2 delete-security-group --group-id "$SECURITY_GROUP_ID"
2>>"$LOG_FILE"; then
               log "Successfully deleted security group"
               break
           else
               ((retry_count++))
               if [[ $retry_count -lt $max_retries ]]; then
                   log "Retry $retry_count/$max_retries: Waiting 10 seconds
before retrying security group deletion..."
                   sleep 10
               else
                   log "ERROR: Failed to delete security group after
$max_retries attempts"
                   ((cleanup_errors++))
               fi
           fi
       done
   fi

   # Delete key pair
   if [[ -n "${KEY_PAIR_NAME:-}" ]]; then
       log "Deleting key pair: $KEY_PAIR_NAME"
       if ! aws ec2 delete-key-pair --key-name "$KEY_PAIR_NAME" 2>>"$LOG_FILE";
then
           log "WARNING: Failed to delete key pair $KEY_PAIR_NAME"
           ((cleanup_errors++))
       fi
       rm -f "${KEY_PAIR_NAME}.pem" 2>>"$LOG_FILE" || log "WARNING: Failed to
remove local key file"
   fi

   # Delete ECS cluster
   if [[ -n "${CLUSTER_NAME:-}" ]]; then
       log "Deleting ECS cluster: $CLUSTER_NAME"
       if ! aws ecs delete-cluster --cluster "$CLUSTER_NAME" 2>>"$LOG_FILE";
then
           log "WARNING: Failed to delete cluster $CLUSTER_NAME"
           ((cleanup_errors++))
```

```
        fi
    fi

    if [[ $cleanup_errors -eq 0 ]]; then
        log "Cleanup completed successfully"
    else
        log "Cleanup completed with $cleanup_errors warnings/errors. Check log
 file for details."
    fi
}

# Function to check prerequisites
check_prerequisites() {
    log "Checking prerequisites..."

    # Check AWS CLI
    if ! command -v aws &> /dev/null; then
        log "ERROR: AWS CLI is not installed"
        exit 1
    fi

    # Check AWS credentials
    if ! aws sts get-caller-identity &> /dev/null; then
        log "ERROR: AWS credentials not configured"
        exit 1
    fi

    # Get caller identity
    CALLER_IDENTITY=$(aws sts get-caller-identity --output text --query
 'Account')
    log "AWS Account: $CALLER_IDENTITY"
    log "AWS Region: $AWS_REGION"

    # Check for default VPC
    DEFAULT_VPC=$(aws ec2 describe-vpcs --filters "Name=is-default,Values=true"
 --query 'Vpcs[0].VpcId' --output text)
    if [[ "$DEFAULT_VPC" == "None" ]]; then
        log "ERROR: No default VPC found. Please create a VPC first."
        exit 1
    fi
    log "Using default VPC: $DEFAULT_VPC"

    # Get default subnet
```

```
    DEFAULT_SUBNET=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=
$DEFAULT_VPC" "Name=default-for-az,Values=true" --query 'Subnets[0].SubnetId' --
output text)
    if [[ "$DEFAULT_SUBNET" == "None" ]]; then
        log "ERROR: No default subnet found"
        exit 1
    fi
    log "Using default subnet: $DEFAULT_SUBNET"

    log "Prerequisites check completed successfully"
}

# Function to create ECS cluster
create_cluster() {
    log "Creating ECS cluster: $CLUSTER_NAME"

    CLUSTER_ARN=$(aws ecs create-cluster --cluster-name "$CLUSTER_NAME" --
tags key=project,value=$PROJECT_TAG key=tutorial,value=$TUTORIAL_TAG --query
 'cluster.clusterArn' --output text)

    if [[ -z "$CLUSTER_ARN" ]]; then
        log "ERROR: Failed to create cluster"
        exit 1
    fi

    log "Created cluster: $CLUSTER_ARN"
    CREATED_RESOURCES+=("ECS Cluster: $CLUSTER_NAME")
}

# Function to create key pair
create_key_pair() {
    log "Creating EC2 key pair: $KEY_PAIR_NAME"

    # FIXED: Set secure umask before key creation
    umask 077
    aws ec2 create-key-pair --key-name "$KEY_PAIR_NAME" --query 'KeyMaterial' --
output text > "${KEY_PAIR_NAME}.pem"
    chmod 400 "${KEY_PAIR_NAME}.pem"
    umask 022  # Reset umask

    log "Created key pair: $KEY_PAIR_NAME"
    CREATED_RESOURCES+=("EC2 Key Pair: $KEY_PAIR_NAME")
```

```
    aws ec2 create-tags --resources "$KEY_PAIR_NAME" --tags Key=project,Value=
$PROJECT_TAG Key=tutorial,Value=$TUTORIAL_TAG 2>>"$LOG_FILE" || log "WARNING:
 Failed to tag key pair"
}

# Function to create security group
create_security_group() {
    log "Creating security group: $SECURITY_GROUP_NAME"

    SECURITY_GROUP_ID=$(aws ec2 create-security-group \
        --group-name "$SECURITY_GROUP_NAME" \
        --description "ECS tutorial security group" \
        --vpc-id "$DEFAULT_VPC" \
        --tag-specifications "ResourceType=security-
group,Tags=[{Key=project,Value=$PROJECT_TAG},{Key=tutorial,Value=$TUTORIAL_TAG}]"
 \
        --query 'GroupId' --output text)

    if [[ -z "$SECURITY_GROUP_ID" ]]; then
        log "ERROR: Failed to create security group"
        exit 1
    fi

    # Add HTTP access rule for nginx web server with restricted CIDR
    # SECURITY FIX: Restrict access to specific CIDR if available, otherwise
 document the risk
    log "WARNING: Security group allows HTTP (port 80) from 0.0.0.0/0 - restrict
 this in production"
    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
        --protocol tcp \
        --port 80 \
        --cidr "0.0.0.0/0"

    log "Created security group: $SECURITY_GROUP_ID"
    log "Added HTTP access on port 80"
    CREATED_RESOURCES+=("Security Group: $SECURITY_GROUP_ID")
}

# Function to get ECS optimized AMI
get_ecs_ami() {
    log "Getting ECS-optimized AMI ID..."

    ECS_AMI_ID=$(aws ssm get-parameters \
```

```
        --names /aws/service/ecs/optimized-ami/amazon-linux-2/recommended \
        --query 'Parameters[0].Value' --output text | jq -r '.image_id')

    if [[ -z "$ECS_AMI_ID" ]]; then
        log "ERROR: Failed to get ECS-optimized AMI ID"
        exit 1
    fi

    log "ECS-optimized AMI ID: $ECS_AMI_ID"
}

# Function to create IAM role for ECS instance (if it doesn't exist)
ensure_ecs_instance_role() {
    log "Checking for ecsInstanceRole..."

    if ! aws iam get-role --role-name ecsInstanceRole &> /dev/null; then
        log "Creating ecsInstanceRole..."

        # Create trust policy
        cat > ecs-instance-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

        # SECURITY FIX: Validate JSON before using
        if ! jq empty ecs-instance-trust-policy.json 2>/dev/null; then
            log "ERROR: Invalid JSON in trust policy"
            rm -f ecs-instance-trust-policy.json
            exit 1
        fi

        # Create role
        aws iam create-role \
            --role-name ecsInstanceRole \
```

```
            --assume-role-policy-document file://ecs-instance-trust-policy.json

        aws iam tag-role --role-name ecsInstanceRole --tags Key=project,Value=
$PROJECT_TAG Key=tutorial,Value=$TUTORIAL_TAG

        # Attach managed policy
        aws iam attach-role-policy \
            --role-name ecsInstanceRole \
            --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role

        # Create instance profile
        aws iam create-instance-profile --instance-profile-name ecsInstanceRole

        # Add role to instance profile
        aws iam add-role-to-instance-profile \
            --instance-profile-name ecsInstanceRole \
            --role-name ecsInstanceRole

        # FIXED: Enhanced wait for role to be ready
        log "Waiting for IAM role to be ready..."
        aws iam wait role-exists --role-name ecsInstanceRole
        sleep 30  # Additional buffer for eventual consistency

        rm -f ecs-instance-trust-policy.json
        log "Created ecsInstanceRole"
        CREATED_RESOURCES+=("IAM Role: ecsInstanceRole")
    else
        log "ecsInstanceRole already exists"
    fi
}

# Function to launch container instance
launch_container_instance() {
    log "Launching ECS container instance..."

    # Create user data script
    cat > ecs-user-data.sh << EOF
#!/bin/bash
echo ECS_CLUSTER=$CLUSTER_NAME >> /etc/ecs/ecs.config
EOF

    # SECURITY FIX: Validate user data script before use
    if ! bash -n ecs-user-data.sh 2>/dev/null; then
```

```
        log "ERROR: Invalid user data script"
        rm -f ecs-user-data.sh
        exit 1
    fi

    INSTANCE_ID=$(aws ec2 run-instances \
        --image-id "$ECS_AMI_ID" \
        --instance-type t3.micro \
        --key-name "$KEY_PAIR_NAME" \
        --security-group-ids "$SECURITY_GROUP_ID" \
        --subnet-id "$DEFAULT_SUBNET" \
        --iam-instance-profile Name=ecsInstanceRole \
        --user-data file://ecs-user-data.sh \
        --tag-specifications "ResourceType=instance,Tags=[{Key=Name,Value=ecs-
tutorial-instance},{Key=project,Value=$PROJECT_TAG},{Key=tutorial,Value=
$TUTORIAL_TAG}]" \
        --monitoring Enabled=false \
        --metadata-options HttpTokens=required,HttpPutResponseHopLimit=1 \
        --query 'Instances[0].InstanceId' --output text)

    if [[ -z "$INSTANCE_ID" ]]; then
        log "ERROR: Failed to launch EC2 instance"
        rm -f ecs-user-data.sh
        exit 1
    fi

    log "Launched EC2 instance: $INSTANCE_ID"
    log "Instance metadata options: IMDSv2 enforced with hop limit 1"
    CREATED_RESOURCES+=("EC2 Instance: $INSTANCE_ID")

    # Wait for instance to be running
    log "Waiting for instance to be running..."
    aws ec2 wait instance-running --instance-ids "$INSTANCE_ID"

    # Wait for ECS agent to register
    log "Waiting for ECS agent to register with cluster..."
    local max_attempts=30
    local attempt=0

    while [[ $attempt -lt $max_attempts ]]; do
        CONTAINER_INSTANCES=$(aws ecs list-container-instances --cluster
 "$CLUSTER_NAME" --query 'containerInstanceArns' --output text)
        if [[ -n "$CONTAINER_INSTANCES" && "$CONTAINER_INSTANCES" != "None" ]];
 then
```

```
            log "Container instance registered successfully"
            break
        fi

        attempt=$((attempt + 1))
        log "Waiting for container instance registration... (attempt $attempt/
$max_attempts)"
        sleep 10
    done

    if [[ $attempt -eq $max_attempts ]]; then
        log "ERROR: Container instance failed to register within expected time"
        rm -f ecs-user-data.sh
        exit 1
    fi

    rm -f ecs-user-data.sh
}

# Function to register task definition
register_task_definition() {
    log "Creating task definition..."

    # Create nginx task definition JSON matching the tutorial
    cat > task-definition.json << 'EOF'
{
    "family": "TASK_FAMILY_PLACEHOLDER",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/docker/library/nginx:latest",
            "cpu": 256,
            "memory": 512,
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
```

```
                    "awslogs-group": "/ecs/nginx-task",
                    "awslogs-region": "REGION_PLACEHOLDER",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ],
    "requiresCompatibilities": ["EC2"],
    "networkMode": "bridge",
    "tags": [
        {
            "key": "project",
            "value": "PROJECT_TAG_PLACEHOLDER"
        },
        {
            "key": "tutorial",
            "value": "TUTORIAL_TAG_PLACEHOLDER"
        }
    ]
}
EOF

    # Replace placeholders securely
    sed -i "s|TASK_FAMILY_PLACEHOLDER|$TASK_FAMILY|g" task-definition.json
    sed -i "s|REGION_PLACEHOLDER|$AWS_REGION|g" task-definition.json
    sed -i "s|PROJECT_TAG_PLACEHOLDER|$PROJECT_TAG|g" task-definition.json
    sed -i "s|TUTORIAL_TAG_PLACEHOLDER|$TUTORIAL_TAG|g" task-definition.json

    # FIXED: Validate JSON before registration
    if ! jq empty task-definition.json 2>/dev/null; then
        log "ERROR: Invalid JSON in task definition"
        rm -f task-definition.json
        exit 1
    fi

    TASK_DEFINITION_ARN=$(aws ecs register-task-definition \
        --cli-input-json file://task-definition.json \
        --query 'taskDefinition.taskDefinitionArn' --output text)

    if [[ -z "$TASK_DEFINITION_ARN" ]]; then
        log "ERROR: Failed to register task definition"
        rm -f task-definition.json
        exit 1
    fi
```

```
    log "Registered task definition: $TASK_DEFINITION_ARN"
    log "Task definition includes CloudWatch Logs configuration for monitoring"
    CREATED_RESOURCES+=("Task Definition: $TASK_DEFINITION_ARN")

    rm -f task-definition.json
}

# Function to create service
create_service() {
    log "Creating ECS service..."

    SERVICE_ARN=$(aws ecs create-service \
        --cluster "$CLUSTER_NAME" \
        --service-name "$SERVICE_NAME" \
        --task-definition "$TASK_FAMILY" \
        --desired-count 1 \
        --tags key=project,value=$PROJECT_TAG key=tutorial,value=$TUTORIAL_TAG \
        --query 'service.serviceArn' --output text)

    if [[ -z "$SERVICE_ARN" ]]; then
        log "ERROR: Failed to create service"
        exit 1
    fi

    log "Created service: $SERVICE_ARN"
    CREATED_RESOURCES+=("ECS Service: $SERVICE_NAME")

    # Wait for service to be stable
    log "Waiting for service to be stable..."
    aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
 "$SERVICE_NAME"

    log "Service is now stable and running"

    # Get the task ARN for monitoring
    TASK_ARN=$(aws ecs list-tasks --cluster "$CLUSTER_NAME" --service-name
 "$SERVICE_NAME" --query 'taskArns[0]' --output text)
    if [[ -n "$TASK_ARN" && "$TASK_ARN" != "None" ]]; then
        log "Service task: $TASK_ARN"
        CREATED_RESOURCES+=("ECS Task: $TASK_ARN")
    fi
}
```

```
# Function to demonstrate monitoring and testing
demonstrate_monitoring() {
    log "Demonstrating monitoring capabilities..."

    # List services
    log "Listing services in cluster:"
    aws ecs list-services --cluster "$CLUSTER_NAME" --output table

    # Describe service
    log "Service details:"
    aws ecs describe-services --cluster "$CLUSTER_NAME" --
services "$SERVICE_NAME" --output table --query 'services[0].
{ServiceName:serviceName,Status:status,RunningCount:runningCount,DesiredCount:desiredCoun

    # List tasks
    log "Listing tasks in service:"
    aws ecs list-tasks --cluster "$CLUSTER_NAME" --service-name "$SERVICE_NAME"
 --output table

    # Describe task
    if [[ -n "$TASK_ARN" && "$TASK_ARN" != "None" ]]; then
        log "Task details:"
        aws ecs describe-tasks --cluster "$CLUSTER_NAME"
 --tasks "$TASK_ARN" --output table --query 'tasks[0].
{TaskArn:taskArn,LastStatus:lastStatus,DesiredStatus:desiredStatus,CreatedAt:createdAt}'
    fi

    # List container instances
    log "Container instances in cluster:"
    aws ecs list-container-instances --cluster "$CLUSTER_NAME" --output table

    # Describe container instance
    CONTAINER_INSTANCE_ARN=$(aws ecs list-container-instances --cluster
 "$CLUSTER_NAME" --query 'containerInstanceArns[0]' --output text)
    if [[ -n "$CONTAINER_INSTANCE_ARN" && "$CONTAINER_INSTANCE_ARN" != "None" ]];
 then
        log "Container instance details:"
        aws ecs describe-container-instances --cluster "$CLUSTER_NAME"
 --container-instances "$CONTAINER_INSTANCE_ARN" --output table --query
 'containerInstances[0].
{Arn:containerInstanceArn,Status:status,RunningTasks:runningTasksCount,PendingTasks:pendi
    fi

    # Test the nginx web server
```

```
    log "Testing nginx web server..."
    PUBLIC_IP=$(aws ec2 describe-instances --instance-ids "$INSTANCE_ID" --query
 'Reservations[0].Instances[0].PublicIpAddress' --output text)

    if [[ -n "$PUBLIC_IP" && "$PUBLIC_IP" != "None" ]]; then
        log "Container instance public IP: $PUBLIC_IP"
        log "Testing HTTP connection to nginx..."

        # Wait a moment for nginx to be fully ready
        sleep 10

        if curl -s --connect-timeout 10 "http://$PUBLIC_IP" | grep -q "Welcome to
 nginx"; then
            log "SUCCESS: Nginx web server is responding correctly"
            echo ""
            echo "=========================================="
            echo "WEB SERVER TEST SUCCESSFUL"
            echo "=========================================="
            echo "You can access your nginx web server at: http://$PUBLIC_IP"
            echo "The nginx welcome page should be visible in your browser."
        else
            log "WARNING: Nginx web server may not be fully ready yet. Try
 accessing http://$PUBLIC_IP in a few minutes."
        fi
    else
        log "WARNING: Could not retrieve public IP address"
    fi
}

# Main execution
main() {
    log "Starting ECS EC2 Launch Type Tutorial (UPDATED VERSION)"
    log "Log file: $LOG_FILE"
    log "Security improvements: IMDSv2 enforced, JSON validation, input
 sanitization, CloudWatch Logs configured"

    check_prerequisites
    create_cluster
    create_key_pair
    create_security_group
    get_ecs_ami
    ensure_ecs_instance_role
    launch_container_instance
    register_task_definition
```

```
    create_service
    demonstrate_monitoring

    log "Tutorial completed successfully!"

    echo ""
    echo "========================================="
    echo "TUTORIAL COMPLETED SUCCESSFULLY"
    echo "========================================="
    echo "Resources created:"
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "  - $resource"
    done
    echo ""
    echo "The nginx service will continue running and maintain the desired task
 count."
    echo "You can monitor the service status using:"
    echo "  aws ecs describe-services --cluster $CLUSTER_NAME --services
 $SERVICE_NAME"
    echo ""
    if [[ -n "${PUBLIC_IP:-}" ]]; then
        echo "Access your web server at: http://$PUBLIC_IP"
        echo ""
    fi
    echo "========================================="
    echo "CLEANUP CONFIRMATION"
    echo "========================================="
    log "Auto-confirming cleanup - proceeding with resource cleanup"
    cleanup_resources
    log "All resources have been cleaned up"
}

# Run main function
main "$@"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddRoleToInstanceProfile
  - AttachRolePolicy
  - AuthorizeSecurityGroupIngress
  - CreateCluster
  - CreateInstanceProfile

- CreateKeyPair
- CreateRole
- CreateSecurityGroup
- CreateService
- DeleteCluster
- DeleteKeyPair
- DeleteSecurityGroup
- DeleteService
- DeregisterTaskDefinition
- DescribeContainerInstances
- DescribeInstances
- DescribeServices
- DescribeSubnets
- DescribeTasks
- DescribeVpcs
- GetCallerIdentity
- GetParameters
- GetRole
- ListContainerInstances
- ListServices
- ListTasks
- RegisterTaskDefinition
- RunInstances
- StopTask
- TerminateInstances
- UpdateService
- Wait (Amazon EC2)
- Wait (Amazon ECS)
- Wait (IAM)

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Creating a managed monitoring workspace

The following code example shows how to:

- Create an IAM role for your workspace
- Create a Grafana workspace
- Configure authentication
- Configure optional settings
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon Managed Grafana Workspace Creation Script
# This script creates an Amazon Managed Grafana workspace and configures it

# Set up logging
LOG_FILE="grafana-workspace-creation.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon Managed Grafana workspace creation script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2
```

```
    if echo "$output" | grep -i "error\|exception\|fail" > /dev/null; then
        echo "ERROR: Command '$cmd' failed with output:"
        echo "$output"
        cleanup_on_error
        exit 1
    fi
}

# Function to clean up resources on error
cleanup_on_error() {
    echo "Error encountered. Attempting to clean up resources..."

    if [ -n "$WORKSPACE_ID" ]; then
        echo "Deleting workspace $WORKSPACE_ID..."
        aws grafana delete-workspace --workspace-id "$WORKSPACE_ID"
    fi

    if [ -n "$ROLE_NAME" ]; then
        echo "Detaching policies from role $ROLE_NAME..."
        if [ -n "$POLICY_ARN" ]; then
            aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$POLICY_ARN"
        fi

        echo "Deleting role $ROLE_NAME..."
        aws iam delete-role --role-name "$ROLE_NAME"
    fi

    if [ -n "$POLICY_ARN" ]; then
        echo "Deleting policy..."
        aws iam delete-policy --policy-arn "$POLICY_ARN"
    fi

    # Clean up JSON files
    rm -f trust-policy.json cloudwatch-policy.json

    echo "Cleanup completed. See $LOG_FILE for details."
}

# Generate a random identifier for resource names
RANDOM_ID=$(openssl rand -hex 4)
WORKSPACE_NAME="GrafanaWorkspace-${RANDOM_ID}"
ROLE_NAME="GrafanaWorkspaceRole-${RANDOM_ID}"
```

```
echo "Using workspace name: $WORKSPACE_NAME"
echo "Using role name: $ROLE_NAME"

# Step 1: Get AWS account ID
echo "Getting AWS account ID..."
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
check_error "$ACCOUNT_ID" "get-caller-identity"
echo "AWS Account ID: $ACCOUNT_ID"

# Step 2: Create IAM role for Grafana workspace
echo "Creating IAM role for Grafana workspace..."

# Create trust policy document
cat > trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "grafana.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role
ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$ROLE_NAME" \
  --assume-role-policy-document file://trust-policy.json \
  --description "Role for Amazon Managed Grafana workspace")

check_error "$ROLE_OUTPUT" "create-role"
aws iam tag-role --role-name "$ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=amazon-managed-grafana-gs
echo "IAM role created successfully"

# Extract role ARN
ROLE_ARN=$(echo "$ROLE_OUTPUT" | grep -o '"Arn": "[^"]*' | cut -d'"' -f4)
echo "Role ARN: $ROLE_ARN"
```

```
# Attach policies to the role
echo "Attaching policies to the role..."

# CloudWatch policy
cat > cloudwatch-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmsForMetric",
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:ListMetrics",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:GetMetricData"
      ],
      "Resource": "*"
    }
  ]
}
EOF

POLICY_OUTPUT=$(aws iam create-policy \
  --policy-name "GrafanaCloudWatchPolicy-${RANDOM_ID}" \
  --policy-document file://cloudwatch-policy.json)

check_error "$POLICY_OUTPUT" "create-policy"

POLICY_ARN=$(echo "$POLICY_OUTPUT" | grep -o '"Arn": "[^"]*' | cut -d'"' -f4)
echo "CloudWatch policy ARN: $POLICY_ARN"

ATTACH_OUTPUT=$(aws iam attach-role-policy \
  --role-name "$ROLE_NAME" \
  --policy-arn "$POLICY_ARN")

check_error "$ATTACH_OUTPUT" "attach-role-policy"
echo "CloudWatch policy attached to role"

# Step 3: Create the Grafana workspace
echo "Creating Amazon Managed Grafana workspace..."
WORKSPACE_OUTPUT=$(aws grafana create-workspace \
  --workspace-name "$WORKSPACE_NAME" \
```

```
  --authentication-providers "SAML" \
  --permission-type "CUSTOMER_MANAGED" \
  --account-access-type "CURRENT_ACCOUNT" \
  --workspace-role-arn "$ROLE_ARN" \
  --workspace-data-sources "CLOUDWATCH" "PROMETHEUS" "XRAY" \
  --grafana-version "10.4" \
  --tags Environment=Development,project=doc-smith,tutorial=amazon-managed-
grafana-gs)

check_error "$WORKSPACE_OUTPUT" "create-workspace"

echo "Workspace creation initiated:"
echo "$WORKSPACE_OUTPUT"

# Extract workspace ID
WORKSPACE_ID=$(echo "$WORKSPACE_OUTPUT" | grep -o '"id": "[^"]*' | cut -d'"' -f4)

if [ -z "$WORKSPACE_ID" ]; then
    echo "ERROR: Failed to extract workspace ID from output"
    exit 1
fi

echo "Workspace ID: $WORKSPACE_ID"

# Step 4: Wait for workspace to become active
echo "Waiting for workspace to become active. This may take several minutes..."
ACTIVE=false
MAX_ATTEMPTS=30
ATTEMPT=0

while [ $ACTIVE = false ] && [ $ATTEMPT -lt $MAX_ATTEMPTS ]; do
    ATTEMPT=$((ATTEMPT+1))
    echo "Checking workspace status (attempt $ATTEMPT of $MAX_ATTEMPTS)..."

    DESCRIBE_OUTPUT=$(aws grafana describe-workspace --workspace-id
 "$WORKSPACE_ID")
    check_error "$DESCRIBE_OUTPUT" "describe-workspace"

    STATUS=$(echo "$DESCRIBE_OUTPUT" | grep -o '"status": "[^"]*' | cut -d'"' -
f4)
    echo "Current status: $STATUS"

    if [ "$STATUS" = "ACTIVE" ]; then
        ACTIVE=true
```

```
        echo "Workspace is now ACTIVE"
    elif [ "$STATUS" = "FAILED" ]; then
        echo "ERROR: Workspace creation failed"
        cleanup_on_error
        exit 1
    else
        echo "Workspace is still being created. Waiting 30 seconds..."
        sleep 30
    fi
done

if [ $ACTIVE = false ]; then
    echo "ERROR: Workspace did not become active within the expected time"
    cleanup_on_error
    exit 1
fi

# Extract workspace endpoint URL
WORKSPACE_URL=$(echo "$DESCRIBE_OUTPUT" | grep -o '"endpoint": "[^"]*' | cut -
d'"' -f4)
echo "Workspace URL: https://$WORKSPACE_URL"

# Step 5: Display workspace information
echo ""
echo "=========================================="
echo "WORKSPACE INFORMATION"
echo "=========================================="
echo "Workspace ID: $WORKSPACE_ID"
echo "Workspace URL: https://$WORKSPACE_URL"
echo "Workspace Name: $WORKSPACE_NAME"
echo "IAM Role: $ROLE_NAME"
echo ""
echo "Note: Since SAML authentication is used, you need to configure SAML
 settings"
echo "using the AWS Management Console or the update-workspace-authentication
 command."
echo "=========================================="

# Step 6: Prompt for cleanup
echo ""
echo "=========================================="
echo "CLEANUP CONFIRMATION"
echo "=========================================="
echo "Resources created:"
```

```
echo "- Amazon Managed Grafana workspace: $WORKSPACE_ID"
echo "- IAM Role: $ROLE_NAME"
echo "- IAM Policy: GrafanaCloudWatchPolicy-${RANDOM_ID}"
echo ""
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "$CLEANUP_CHOICE" =~ ^[Yy] ]]; then
    echo "Cleaning up resources..."

    echo "Deleting workspace $WORKSPACE_ID..."
    DELETE_OUTPUT=$(aws grafana delete-workspace --workspace-id "$WORKSPACE_ID")
    check_error "$DELETE_OUTPUT" "delete-workspace"

    echo "Waiting for workspace to be deleted..."
    DELETED=false
    ATTEMPT=0

    while [ $DELETED = false ] && [ $ATTEMPT -lt $MAX_ATTEMPTS ]; do
        ATTEMPT=$((ATTEMPT+1))
        echo "Checking deletion status (attempt $ATTEMPT of $MAX_ATTEMPTS)..."

        if aws grafana describe-workspace --workspace-id "$WORKSPACE_ID" 2>&1 |
 grep -i "not found\|does not exist" > /dev/null; then
            DELETED=true
            echo "Workspace has been deleted"
        else
            echo "Workspace is still being deleted. Waiting 30 seconds..."
            sleep 30
        fi
    done

    if [ $DELETED = false ]; then
        echo "WARNING: Workspace deletion is taking longer than expected. It may
 still be in progress."
    fi

    # Detach policy from role
    echo "Detaching policy from role..."
    aws iam detach-role-policy \
      --role-name "$ROLE_NAME" \
      --policy-arn "$POLICY_ARN"

    # Delete policy
```

```
    echo "Deleting IAM policy..."
    aws iam delete-policy \
      --policy-arn "$POLICY_ARN"

    # Delete role
    echo "Deleting IAM role..."
    aws iam delete-role \
      --role-name "$ROLE_NAME"

    # Clean up JSON files
    rm -f trust-policy.json cloudwatch-policy.json

    echo "Cleanup completed"
else
    echo "Skipping cleanup. Resources will remain in your AWS account."
fi

echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreatePolicy
  - CreateRole
  - CreateWorkspace
  - DeletePolicy
  - DeleteRole
  - DeleteWorkspace
  - DescribeWorkspace
  - DetachRolePolicy
  - GetCallerIdentity

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Creating a monitoring dashboard with function name as a variable

The following code example shows how to:

- Create a CloudWatch dashboard
- Add Lambda metrics widgets with a function name variable
- Verify the dashboard
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Script to create a CloudWatch dashboard with Lambda function name as a variable
# This script creates a CloudWatch dashboard that allows you to switch between
 different Lambda functions

# Set up logging with secure permissions
LOG_FILE="${HOME}/.cloudwatch-dashboard-script.log"
touch "$LOG_FILE" && chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "$(date): Starting CloudWatch dashboard creation script"

# Security: Set strict error handling
set -uo pipefail
trap 'handle_error "Script failed at line $LINENO"' ERR

# Function to handle errors
handle_error() {
    local error_msg="${1:-Unknown error}"
    echo "ERROR: $error_msg" >&2
```

```
    echo "Resources created:"
    echo "- CloudWatch Dashboard: LambdaMetricsDashboard"
    echo ""
    echo "=========================================="
    echo "CLEANUP CONFIRMATION"
    echo "=========================================="
    echo "An error occurred. Proceeding with automatic cleanup..."

    echo "Cleaning up resources..."
    aws cloudwatch delete-dashboards --dashboard-names LambdaMetricsDashboard 2>/
dev/null || true

    # Clean up temporary files
    if [ -n "${TEMP_DIR:-}" ] && [ -d "$TEMP_DIR" ]; then
        rm -rf "$TEMP_DIR"
    fi
    rm -f dashboard-body.json

    echo "Cleanup complete."
    exit 1
}

# Security: Validate AWS CLI is installed
if ! command -v aws &> /dev/null; then
    handle_error "AWS CLI is not installed. Please install it and try again."
fi

# Check if AWS CLI is installed and configured
echo "Checking AWS CLI configuration..."
if ! aws sts get-caller-identity > /dev/null 2>&1; then
    handle_error "AWS CLI is not properly configured. Please configure it with
 'aws configure' and try again."
fi

# Get the current region securely
REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$REGION" ]; then
    REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $REGION"
fi
echo "Using region: $REGION"

# Validate region format
if ! [[ "$REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]{1}$ ]]; then
```

```
    handle_error "Invalid AWS region format: $REGION"
fi

# Check if there are any Lambda functions in the account
echo "Checking for Lambda functions..."
LAMBDA_FUNCTIONS=$(aws lambda list-functions --region "$REGION" --query
 "Functions[*].FunctionName" --output text 2>/dev/null || echo "")

if [ -z "$LAMBDA_FUNCTIONS" ]; then
    echo "No Lambda functions found in your account. Creating a simple test
 function..."

    # Create a temporary directory for Lambda function code with secure
 permissions
    TEMP_DIR=$(mktemp -d)
    chmod 700 "$TEMP_DIR"
    trap 'rm -rf "$TEMP_DIR"' EXIT

    # Create a simple Lambda function
    cat > "$TEMP_DIR/index.js" << 'EOF'
exports.handler = async (event) => {
    console.log('Event:', JSON.stringify(event, null, 2));
    return {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
};
EOF

    # Zip the function code
    if ! cd "$TEMP_DIR"; then
        handle_error "Failed to change to temporary directory"
    fi

    if ! zip -q function.zip index.js; then
        handle_error "Failed to create zip file"
    fi

    # Create a role for the Lambda function with restricted trust policy
    ROLE_NAME="LambdaDashboardTestRole-$(date +%s)"
    TRUST_POLICY='{"Version":"2012-10-17",       "Statement":
[{"Effect":"Allow","Principal":
{"Service":"lambda.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
```

```
    if ! ROLE_ARN=$(aws iam create-role \
        --role-name "$ROLE_NAME" \
        --assume-role-policy-document "$TRUST_POLICY" \
        --tags Key=project,Value=doc-smith Key=tutorial,Value=cloudwatch-
dynamicdash \
        --query "Role.Arn" \
        --output text 2>/dev/null); then
        handle_error "Failed to create IAM role for Lambda function"
    fi

    echo "Waiting for role to be available..."
    sleep 10

    # Attach basic Lambda execution policy
    if ! aws iam attach-role-policy \
        --role-name "$ROLE_NAME" \
        --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"; then
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to attach policy to IAM role"
    fi

    # Create the Lambda function
    FUNCTION_NAME="DashboardTestFunction-$(date +%s)"
    if ! aws lambda create-function \
        --function-name "$FUNCTION_NAME" \
        --runtime nodejs18.x \
        --role "$ROLE_ARN" \
        --handler index.handler \
        --zip-file fileb://function.zip \
        --tags project=doc-smith,tutorial=cloudwatch-dynamicdash \
        --region "$REGION" > /dev/null 2>&1; then
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to create Lambda function"
    fi

    # Invoke the function to generate some metrics
    echo "Invoking Lambda function to generate metrics..."
    for i in {1..5}; do
```

```
        aws lambda invoke --function-name "$FUNCTION_NAME" --payload '{}' /dev/
null --region "$REGION" > /dev/null 2>&1 || true
        sleep 1
    done

    # Go back to original directory
    cd - > /dev/null

    # Set the function name for the dashboard
    DEFAULT_FUNCTION="$FUNCTION_NAME"
else
    # Use the first Lambda function as default
    DEFAULT_FUNCTION=$(echo "$LAMBDA_FUNCTIONS" | awk '{print $1}')
    echo "Found Lambda functions. Using $DEFAULT_FUNCTION as default."
    FUNCTION_NAME=""
    ROLE_NAME=""
fi

# Create a dashboard with Lambda metrics and a function name variable
echo "Creating CloudWatch dashboard with Lambda function name variable..."

# Create a JSON file for the dashboard body with secure permissions
DASHBOARD_JSON="dashboard-body-$$.json"
touch "$DASHBOARD_JSON" && chmod 600 "$DASHBOARD_JSON"

# Escape special characters in region and function name for JSON
REGION_ESCAPED=$(printf '%s\n' "$REGION" | sed 's:[\/&]:\\&:g')
FUNCTION_ESCAPED=$(printf '%s\n' "$DEFAULT_FUNCTION" | sed 's:[\/&]:\\&:g')

cat > "$DASHBOARD_JSON" << EOF
{
  "widgets": [
    {
      "type": "metric",
      "x": 0,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "\${FunctionName}" ],
          [ ".", "Errors", ".", "." ],
          [ ".", "Throttles", ".", "." ]
        ],
```

```
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Lambda Function Metrics for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 0,
      "y": 6,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Duration", "FunctionName", "\${FunctionName}",
 { "stat": "Average" } ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Duration for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 12,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "ConcurrentExecutions", "FunctionName", "\
${FunctionName}" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Concurrent Executions for \${FunctionName}",
        "period": 300
      }
    }
  ],
```

```
  "periodOverride": "auto",
  "variables": [
    {
      "type": "property",
      "id": "FunctionName",
      "property": "FunctionName",
      "label": "Lambda Function",
      "inputType": "select",
      "values": [
        {
          "value": "$FUNCTION_ESCAPED",
          "label": "$FUNCTION_ESCAPED"
        }
      ]
    }
  ]
}
EOF

# Validate JSON before sending
if ! jq empty "$DASHBOARD_JSON" 2>/dev/null; then
    handle_error "Invalid JSON generated for dashboard"
fi

# Create the dashboard using the JSON file
DASHBOARD_NAME="LambdaMetricsDashboard-$(date +%s)"
if ! DASHBOARD_RESULT=$(aws cloudwatch put-dashboard \
    --dashboard-name "$DASHBOARD_NAME" \
    --dashboard-body file://"$DASHBOARD_JSON" \
    --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to create CloudWatch dashboard."
fi

# Display any validation messages but continue
```

```
if echo "$DASHBOARD_RESULT" | grep -q "DashboardValidationMessages"; then
    echo "Dashboard created with validation messages:"
    echo "$DASHBOARD_RESULT"
    echo "These validation messages are warnings and the dashboard should still
 function."
else
    echo "Dashboard created successfully!"
fi

# Verify the dashboard was created
echo "Verifying dashboard creation..."
if ! DASHBOARD_INFO=$(aws cloudwatch get-dashboard --dashboard-name
 "$DASHBOARD_NAME" --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to verify dashboard creation."
fi

echo "Dashboard verification successful!"
echo "Dashboard details:"
echo "$DASHBOARD_INFO"

# List all dashboards to confirm
echo "Listing all dashboards:"
if ! DASHBOARDS=$(aws cloudwatch list-dashboards --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to list dashboards."
```

```
fi
echo "$DASHBOARDS"

# Show instructions for accessing the dashboard
echo ""
echo "Dashboard created successfully! To access it:"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/"
echo "2. In the navigation pane, choose Dashboards"
echo "3. Select $DASHBOARD_NAME"
echo "4. You should see a dropdown menu labeled 'Lambda Function' at the top of
 the dashboard"
echo "5. Use this dropdown to select different Lambda functions and see their
 metrics"
echo ""

# Create a list of resources for cleanup
RESOURCES=("- CloudWatch Dashboard: $DASHBOARD_NAME")
if [ -n "${FUNCTION_NAME:-}" ]; then
    RESOURCES+=("- Lambda Function: $FUNCTION_NAME")
    RESOURCES+=("- IAM Role: $ROLE_NAME")
fi

# Prompt for cleanup with automatic yes
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Resources created:"
for resource in "${RESOURCES[@]}"; do
    echo "$resource"
done
echo ""
echo "Proceeding with automatic cleanup..."

CLEANUP_CHOICE="y"

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    echo "Cleaning up resources..."

    # Delete the dashboard
    if aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" --
region "$REGION" 2>/dev/null; then
        echo "Dashboard deleted successfully."
    else
```

```
        echo "WARNING: Failed to delete dashboard. You may need to delete it
 manually."
    fi

    # If we created a Lambda function, delete it and its role
    if [ -n "${FUNCTION_NAME:-}" ]; then
        echo "Deleting Lambda function..."
        if aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null; then
            echo "Lambda function deleted successfully."
        else
            echo "WARNING: Failed to delete Lambda function. You may need to
 delete it manually."
        fi

        echo "Detaching role policy..."
        if aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null; then
            echo "Role policy detached successfully."
        else
            echo "WARNING: Failed to detach role policy. You may need to detach
 it manually."
        fi

        echo "Deleting IAM role..."
        if aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null; then
            echo "IAM role deleted successfully."
        else
            echo "WARNING: Failed to delete IAM role. You may need to delete it
 manually."
        fi
    fi

    # Clean up the JSON file
    rm -f "$DASHBOARD_JSON"

    echo "Cleanup complete."
fi

echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.

- AttachRolePolicy
- CreateFunction
- CreateRole
- DeleteDashboards
- DeleteFunction
- DeleteRole
- DetachRolePolicy
- GetCallerIdentity
- GetDashboard
- Invoke
- ListDashboards
- ListFunctions
- PutDashboard

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Creating your first serverless function

The following code example shows how to:

- Create an IAM role for Lambda
- Create function code
- Create a Lambda function
- Test your Lambda function
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash
# AWS Lambda - Create Your First Function
# This script creates a Lambda function, invokes it with a test event,
# views CloudWatch logs, and cleans up all resources.
#
# Source: https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html
#
# Resources created:
#   - IAM role (Lambda execution role with basic logging permissions)
#   - Lambda function (Python 3.13 or Node.js 22.x runtime)
#   - CloudWatch log group (created automatically by Lambda on invocation)

set -eE -o pipefail

###############################################################################
# Setup
###############################################################################

UNIQUE_ID=$(head -c 8 /dev/urandom | od -An -tx1 | tr -d ' ')
FUNCTION_NAME="my-lambda-function-${UNIQUE_ID}"
ROLE_NAME="lambda-execution-role-${UNIQUE_ID}"
LOG_GROUP_NAME="/aws/lambda/${FUNCTION_NAME}"

TEMP_DIR=$(mktemp -d)
readonly TEMP_DIR
LOG_FILE="${TEMP_DIR}/lambda-gettingstarted.log"

exec > >(tee -a "$LOG_FILE") 2>&1

declare -a CREATED_RESOURCES

###############################################################################
```

```
# Helper functions
################################################################################

cleanup_resources() {
    # Disable error trap to prevent recursion during cleanup
    trap - ERR
    set +eE

    echo ""
    echo "Cleaning up resources..."
    echo ""

    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        local RESOURCE="${CREATED_RESOURCES[$i]}"
        local TYPE="${RESOURCE%%:*}"
        local NAME="${RESOURCE#*:}"

        case "$TYPE" in
            log-group)
                echo "Deleting CloudWatch log group: ${NAME}"
                aws logs delete-log-group \
                    --log-group-name "$NAME" 2>&1 || echo "  WARNING: Could not
 delete log group ${NAME}."
                ;;
            lambda-function)
                echo "Deleting Lambda function: ${NAME}"
                aws lambda delete-function \
                    --function-name "$NAME" 2>&1 || echo "  WARNING: Could not
 delete Lambda function ${NAME}."
                echo "  Waiting for function deletion to complete..."
                local DELETE_WAIT=0
                while aws lambda get-function --function-name "$NAME" > /dev/null
 2>&1; do
                    sleep 2
                    DELETE_WAIT=$((DELETE_WAIT + 2))
                    if [ "$DELETE_WAIT" -ge 60 ]; then
                        echo "  WARNING: Timed out waiting for function
 deletion."
                        break
                    fi
                done
                ;;
            iam-role-policy)
                local ROLE_PART="${NAME%%|*}"
```

```
                local POLICY_PART="${NAME#*|}"
                echo "Detaching policy from role: ${ROLE_PART}"
                aws iam detach-role-policy \
                    --role-name "$ROLE_PART" \
                    --policy-arn "$POLICY_PART" 2>&1 || echo "  WARNING: Could
 not detach policy from role ${ROLE_PART}."
                ;;
            iam-role)
                echo "Deleting IAM role: ${NAME}"
                aws iam delete-role \
                    --role-name "$NAME" 2>&1 || echo "  WARNING: Could not delete
 IAM role ${NAME}."
                ;;
        esac
    done

    if [ -d "$TEMP_DIR" ]; then
        rm -rf "$TEMP_DIR"
    fi

    echo ""
    echo "Cleanup complete."
}

handle_error() {
    echo ""
    echo "========================================="
    echo "ERROR: Script failed at $1"
    echo "========================================="
    echo ""
    if [ ${#CREATED_RESOURCES[@]} -gt 0 ]; then
        echo "Attempting to clean up ${#CREATED_RESOURCES[@]} resource(s)..."
        cleanup_resources
    fi
    exit 1
}

trap 'handle_error "line $LINENO"' ERR

wait_for_resource() {
    local DESCRIPTION="$1"
    local COMMAND="$2"
    local TARGET_VALUE="$3"
    local TIMEOUT=300
```

```
    local ELAPSED=0
    local INTERVAL=5

    echo "Waiting for ${DESCRIPTION}..."
    while true; do
        local RESULT
        RESULT=$(eval "$COMMAND" 2>&1) || true
        if echo "$RESULT" | grep -q "$TARGET_VALUE"; then
            echo "  ${DESCRIPTION} is ready."
            return 0
        fi
        if [ "$ELAPSED" -ge "$TIMEOUT" ]; then
            echo "ERROR: Timed out waiting for ${DESCRIPTION} after ${TIMEOUT}
 seconds."
            return 1
        fi
        sleep "$INTERVAL"
        ELAPSED=$((ELAPSED + INTERVAL))
    done
}

validate_input() {
    local input="$1"
    local pattern="$2"
    if ! [[ "$input" =~ $pattern ]]; then
        echo "ERROR: Invalid input: $input"
        return 1
    fi
    return 0
}

################################################################################
# Region pre-check
################################################################################

CONFIGURED_REGION=$(aws configure get region 2>/dev/null || true)
if [ -z "$CONFIGURED_REGION" ] && [ -z "$AWS_DEFAULT_REGION" ] && [ -z
 "$AWS_REGION" ]; then
    echo "ERROR: No AWS region configured."
    echo "Run 'aws configure set region <region>' or export AWS_DEFAULT_REGION."
    exit 1
fi

################################################################################
```

```
# Runtime selection
################################################################################

echo ""
echo "=========================================="
echo "AWS Lambda - Create Your First Function"
echo "=========================================="
echo ""
echo "Select a runtime for your Lambda function:"
echo "  1) Python 3.13"
echo "  2) Node.js 22.x"
echo ""
echo "Using default: Python 3.13"
RUNTIME_CHOICE="1"

case "$RUNTIME_CHOICE" in
    1)
        RUNTIME="python3.13"
        HANDLER="lambda_function.lambda_handler"
        CODE_FILE="lambda_function.py"
        cat > "${TEMP_DIR}/${CODE_FILE}" << 'PYTHON_EOF'
import json
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    if not isinstance(event, dict) or 'length' not in event or 'width' not in
 event:
        raise ValueError('Event must contain length and width')
    try:
        length = float(event['length'])
        width = float(event['width'])
        if length < 0 or width < 0:
            raise ValueError('Length and width must be non-negative')
        area = calculate_area(length, width)
        print(f'The area is {area}')
        logger.info(f'CloudWatch logs group: {context.log_group_name}')
        return json.dumps({'area': area})
    except (TypeError, ValueError) as e:
        logger.error(f'Error processing input: {str(e)}')
        raise
def calculate_area(length, width):
    return length * width
PYTHON_EOF
```

```
        echo "Selected runtime: Python 3.13"
        ;;
    2)
        RUNTIME="nodejs22.x"
        HANDLER="index.handler"
        CODE_FILE="index.mjs"
        cat > "${TEMP_DIR}/${CODE_FILE}" << 'NODEJS_EOF'
export const handler = async (event, context) => {
  if (!event || typeof event.length !== 'number' || typeof event.width !==
 'number') {
    throw new Error('Event must contain numeric length and width');
  }
  if (event.length < 0 || event.width < 0) {
    throw new Error('Length and width must be non-negative');
  }
  const area = event.length * event.width;
  console.log(`The area is ${area}`);
  console.log('CloudWatch log group: ', context.logGroupName);
  return JSON.stringify({area});
};
NODEJS_EOF
        echo "Selected runtime: Node.js 22.x"
        ;;
    *)
        echo "ERROR: Invalid choice. Please enter 1 or 2."
        exit 1
        ;;
esac

# ============================================================================
# Step 1: Create IAM execution role
# ============================================================================

echo ""
echo "=========================================="
echo "Step 1: Create IAM execution role"
echo "=========================================="
echo ""

TRUST_POLICY='{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'

echo "Creating IAM role: ${ROLE_NAME}"
ROLE_OUTPUT=$(aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document "$TRUST_POLICY" \
    --query 'Role.Arn' \
    --output text 2>&1)

if ! validate_input "$ROLE_OUTPUT" "^arn:aws:iam::[0-9]+:role/"; then
    echo "ERROR: Failed to create IAM role"
    exit 1
fi

echo "$ROLE_OUTPUT"
ROLE_ARN="$ROLE_OUTPUT"
CREATED_RESOURCES+=("iam-role:${ROLE_NAME}")
echo "Role ARN: ${ROLE_ARN}"

aws iam tag-role \
    --role-name "$ROLE_NAME" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=lambda-gettingstarted

echo ""
echo "Attaching AWSLambdaBasicExecutionRole policy..."
aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>&1
CREATED_RESOURCES+=("iam-role-policy:${ROLE_NAME}|arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole")
echo "Policy attached."

# IAM roles can take a few seconds to propagate
echo "Waiting for IAM role to propagate..."
sleep 10

################################################################################
```

```
# Step 2: Create Lambda function
################################################################################

echo ""
echo "=========================================="
echo "Step 2: Create Lambda function"
echo "=========================================="
echo ""

echo "Creating deployment package..."
ORIGINAL_DIR=$(pwd)
cd "$TEMP_DIR" || exit 1
zip -j function.zip "$CODE_FILE" > /dev/null 2>&1 || {
    echo "ERROR: Failed to create deployment package"
    exit 1
}
cd "$ORIGINAL_DIR" || exit 1

if [ ! -f "${TEMP_DIR}/function.zip" ]; then
    echo "ERROR: Deployment package creation failed"
    exit 1
fi

echo "Creating Lambda function: ${FUNCTION_NAME}"
echo "  Runtime: ${RUNTIME}"
echo "  Handler: ${HANDLER}"
echo ""

CREATE_OUTPUT=$(aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime "$RUNTIME" \
    --role "$ROLE_ARN" \
    --handler "$HANDLER" \
    --architectures x86_64 \
    --zip-file "fileb://${TEMP_DIR}/function.zip" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted \
    --query '[FunctionName, FunctionArn, Runtime, State]' \
    --output text 2>&1)

if [ -z "$CREATE_OUTPUT" ]; then
    echo "ERROR: Failed to create Lambda function"
    exit 1
fi
```

```
echo "$CREATE_OUTPUT"
CREATED_RESOURCES+=("lambda-function:${FUNCTION_NAME}")

wait_for_resource "Lambda function to become Active" \
    "aws lambda get-function-configuration --function-name ${FUNCTION_NAME} --
query State --output text" \
    "Active"

################################################################################
# Step 3: Invoke the function
################################################################################

echo ""
echo "========================================="
echo "Step 3: Invoke the function"
echo "========================================="
echo ""

TEST_EVENT='{"length": 6, "width": 7}'
echo "Invoking function with test event: ${TEST_EVENT}"
echo ""

echo "$TEST_EVENT" > "${TEMP_DIR}/test-event.json"

if ! validate_input "$TEST_EVENT" '"length": [0-9]+, "width": [0-9]+'; then
    echo "ERROR: Invalid test event format"
    exit 1
fi

INVOKE_OUTPUT=$(aws lambda invoke \
    --function-name "$FUNCTION_NAME" \
    --payload "fileb://${TEMP_DIR}/test-event.json" \
    --cli-read-timeout 30 \
    "${TEMP_DIR}/response.json" 2>&1)
echo "$INVOKE_OUTPUT"

if [ ! -f "${TEMP_DIR}/response.json" ]; then
    echo "ERROR: No response file generated"
    exit 1
fi

RESPONSE=$(cat "${TEMP_DIR}/response.json")
echo ""
echo "Function response: ${RESPONSE}"
```

```
echo ""

if echo "$INVOKE_OUTPUT" | grep -qi "functionerror"; then
    echo "WARNING: Function returned an error."
fi

################################################################################
# Step 4: View CloudWatch logs
################################################################################

echo ""
echo "============================================"
echo "Step 4: View CloudWatch Logs"
echo "============================================"
echo ""

echo "Log group: ${LOG_GROUP_NAME}"
echo ""

echo "Waiting for CloudWatch logs to be available..."

LOG_STREAMS=""
for i in $(seq 1 6); do
    LOG_STREAMS=$(aws logs describe-log-streams \
        --log-group-name "$LOG_GROUP_NAME" \
        --order-by LastEventTime \
        --descending \
        --query 'logStreams[0].logStreamName' \
        --output text 2>/dev/null) || true
    if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
        break
    fi
    LOG_STREAMS=""
    sleep 5
done

if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
    echo "Latest log stream: ${LOG_STREAMS}"
    echo ""
    echo "--- Log events ---"
    LOG_EVENTS=$(aws logs get-log-events \
        --log-group-name "$LOG_GROUP_NAME" \
        --log-stream-name "$LOG_STREAMS" \
        --query 'events[].message' \
```

```
        --output text 2>&1) || true
    echo "$LOG_EVENTS"
    echo "--- End of log events ---"
else
    echo "No log streams found yet. Logs may take a moment to appear."
    echo "You can view them in the CloudWatch console:"
    echo "  Log group: ${LOG_GROUP_NAME}"
fi

CREATED_RESOURCES+=("log-group:${LOG_GROUP_NAME}")

aws logs tag-log-group \
    --log-group-name "$LOG_GROUP_NAME" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted

################################################################################
# Summary and cleanup
################################################################################

echo ""
echo "==========================================="
echo "SUMMARY"
echo "==========================================="
echo ""
echo "Resources created:"
echo "  IAM role:          ${ROLE_NAME}"
echo "  Lambda function:   ${FUNCTION_NAME}"
echo "  CloudWatch logs:   ${LOG_GROUP_NAME}"
echo ""
echo "==========================================="
echo "CLEANUP"
echo "==========================================="
echo ""
echo "Cleaning up all created resources..."
cleanup_resources

echo ""
echo "Done."
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateFunction

- CreateRole
- DeleteFunction
- DeleteLogGroup
- DeleteRole
- DescribeLogStreams
- DetachRolePolicy
- GetFunction
- GetFunctionConfiguration
- GetLogEvents
- Invoke

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get started with serverless data warehouses using the CLI

The following code example shows how to:

- Use secrets-manager CreateSecret
- Use secrets-manager DeleteSecret
- Use secrets-manager GetSecretValue
- Use redshift CreateNamespace
- Use redshift CreateWorkgroup
- Use redshift DeleteNamespace
- Use iam CreateRole

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon Redshift Serverless Tutorial Script with Secrets Manager (No jq
 dependency)
# This script creates a Redshift Serverless environment, loads sample data, and
 runs queries
# Uses AWS Secrets Manager for secure password management without requiring jq

# Set up logging
LOG_FILE="redshift-serverless-tutorial-v4.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon Redshift Serverless tutorial script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Function to check for errors in command output
check_error() {
  local output=$1
  local cmd=$2

  if echo "$output" | grep -i "error\|exception\|fail" > /dev/null; then
    echo "ERROR: Command failed: $cmd"
    echo "Output: $output"
    cleanup_resources
    exit 1
  fi
}

# Function to generate a secure password that meets Redshift requirements
generate_secure_password() {
  # Redshift password requirements:
  # - 8-64 characters
```

```
  # - At least one uppercase letter
  # - At least one lowercase letter
  # - At least one decimal digit
  # - Can contain printable ASCII characters except /, ", ', \, @, space

  local password=""
  local valid=false
  local attempts=0
  local max_attempts=10

  while [[ "$valid" == false && $attempts -lt $max_attempts ]]; do
    # Generate base password with safe characters
    local base=$(openssl rand -base64 12 | tr -d '/+=' | head -c 12)

    # Ensure we have at least one of each required character type
    local upper=$(echo "ABCDEFGHIJKLMNOPQRSTUVWXYZ" | fold -w1 | shuf -n1)
    local lower=$(echo "abcdefghijklmnopqrstuvwxyz" | fold -w1 | shuf -n1)
    local digit=$(echo "0123456789" | fold -w1 | shuf -n1)
    local special=$(echo "!#$%&*()_+-=[]{}|;:,.<>?" | fold -w1 | shuf -n1)

    # Combine and shuffle
    password="${base}${upper}${lower}${digit}${special}"
    password=$(echo "$password" | fold -w1 | shuf | tr -d '\n')

    # Validate password meets requirements
    if [[ ${#password} -ge 8 && ${#password} -le 64 ]] && \
       [[ "$password" =~ [A-Z] ]] && \
       [[ "$password" =~ [a-z] ]] && \
       [[ "$password" =~ [0-9] ]] && \
       [[ ! "$password" =~ [/\"\'\\@[:space:]] ]]; then
      valid=true
    fi

    ((attempts++))
  done

  if [[ "$valid" == false ]]; then
    echo "ERROR: Failed to generate valid password after $max_attempts attempts"
    exit 1
  fi

  echo "$password"
}
```

```
# Function to create secret in AWS Secrets Manager
create_secret() {
  local secret_name=$1
  local username=$2
  local password=$3
  local description=$4

  echo "Creating secret in AWS Secrets Manager: $secret_name"

  # Create the secret using AWS CLI without jq
  local secret_output=$(aws secretsmanager create-secret \
    --name "$secret_name" \
    --description "$description" \
    --secret-string "{\"username\":\"$username\",\"password\":\"$password\"}" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=redshift-serverless
 2>&1)

  if echo "$secret_output" | grep -i "error\|exception\|fail" > /dev/null; then
    echo "ERROR: Failed to create secret: $secret_output"
    return 1
  fi

  echo "Secret created successfully: $secret_name"
  return 0
}

# Function to retrieve password from AWS Secrets Manager
get_password_from_secret() {
  local secret_name=$1

  # Get the secret value and extract password using sed/grep instead of jq
  local secret_value=$(aws secretsmanager get-secret-value \
    --secret-id "$secret_name" \
    --query 'SecretString' \
    --output text 2>/dev/null)

  if [[ $? -eq 0 ]]; then
    # Extract password from JSON using sed
    echo "$secret_value" | sed -n 's/.*"password":"\([^"]*\)".*/\1/p'
  else
    echo ""
  fi
}
```

```
# Function to wait for a resource to be available
wait_for_resource() {
  local resource_type=$1
  local resource_name=$2
  local max_attempts=$3
  local wait_seconds=$4
  local check_cmd=$5

  echo "Waiting for $resource_type $resource_name to be available..."

  for ((i=1; i<=$max_attempts; i++)); do
    local output=$($check_cmd 2>/dev/null)
    local status=$(echo "$output" | grep -o '"Status": "[^"]*' | cut -d'"' -f4 ||
 echo "")

    if [[ "$status" == "AVAILABLE" ]]; then
      echo "$resource_type $resource_name is now available"
      return 0
    fi

    echo "Attempt $i/$max_attempts: $resource_type $resource_name status:
 $status. Waiting $wait_seconds seconds..."
    sleep $wait_seconds
  done

  echo "ERROR: Timed out waiting for $resource_type $resource_name to be
 available"
  return 1
}

# Function to wait for a resource to be deleted
wait_for_resource_deletion() {
  local resource_type=$1
  local resource_name=$2
  local max_attempts=$3
  local wait_seconds=$4
  local check_cmd=$5

  echo "Waiting for $resource_type $resource_name to be deleted..."

  for ((i=1; i<=$max_attempts; i++)); do
    local output=$($check_cmd 2>&1)

    if echo "$output" | grep -i "not found\|does not exist" > /dev/null; then
```

```
      echo "$resource_type $resource_name has been deleted"
      return 0
    fi

    echo "Attempt $i/$max_attempts: $resource_type $resource_name is still being
 deleted. Waiting $wait_seconds seconds..."
    sleep $wait_seconds
  done

  echo "ERROR: Timed out waiting for $resource_type $resource_name to be deleted"
  return 1
}

# Function to clean up resources
cleanup_resources() {
  echo ""
  echo "========================================="
  echo "CLEANUP CONFIRMATION"
  echo "========================================="
  echo "The following resources were created:"
  echo "- Redshift Serverless Workgroup: $WORKGROUP_NAME"
  echo "- Redshift Serverless Namespace: $NAMESPACE_NAME"
  echo "- IAM Role: $ROLE_NAME"
  echo "- Secrets Manager Secret: $SECRET_NAME"
  echo ""
  echo "Do you want to clean up all created resources? (y/n): "
  read -r CLEANUP_CHOICE

  if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    echo "Cleaning up resources..."

    # Delete the workgroup
    echo "Deleting Redshift Serverless workgroup $WORKGROUP_NAME..."
    WORKGROUP_DELETE_OUTPUT=$(aws redshift-serverless delete-workgroup --
workgroup-name "$WORKGROUP_NAME" 2>&1)
    echo "$WORKGROUP_DELETE_OUTPUT"

    # Wait for workgroup to be deleted before deleting namespace
    wait_for_resource_deletion "workgroup" "$WORKGROUP_NAME" 20 30 "aws redshift-
serverless get-workgroup --workgroup-name $WORKGROUP_NAME"

    # Delete the namespace
    echo "Deleting Redshift Serverless namespace $NAMESPACE_NAME..."
```

```
    NAMESPACE_DELETE_OUTPUT=$(aws redshift-serverless delete-namespace --
namespace-name "$NAMESPACE_NAME" 2>&1)
    echo "$NAMESPACE_DELETE_OUTPUT"

    # Wait for namespace to be deleted
    wait_for_resource_deletion "namespace" "$NAMESPACE_NAME" 20 30 "aws redshift-
serverless get-namespace --namespace-name $NAMESPACE_NAME"

    # Delete the IAM role policy
    echo "Deleting IAM role policy..."
    POLICY_DELETE_OUTPUT=$(aws iam delete-role-policy --role-name "$ROLE_NAME" --
policy-name S3Access 2>&1)
    echo "$POLICY_DELETE_OUTPUT"

    # Delete the IAM role
    echo "Deleting IAM role $ROLE_NAME..."
    ROLE_DELETE_OUTPUT=$(aws iam delete-role --role-name "$ROLE_NAME" 2>&1)
    echo "$ROLE_DELETE_OUTPUT"

    # Delete the secret
    echo "Deleting Secrets Manager secret $SECRET_NAME..."
    SECRET_DELETE_OUTPUT=$(aws secretsmanager delete-secret --secret-id
 "$SECRET_NAME" --force-delete-without-recovery 2>&1)
    echo "$SECRET_DELETE_OUTPUT"

    echo "Cleanup completed."
  else
    echo "Cleanup skipped. Resources will remain in your AWS account."
  fi
}

# Check if required tools are available
if ! command -v openssl &> /dev/null; then
    echo "ERROR: openssl is required but not installed. Please install openssl to
 continue."
    exit 1
fi

# Generate unique names for resources
RANDOM_SUFFIX=$(cat /dev/urandom | tr -dc 'a-z0-9' | head -c 6)
NAMESPACE_NAME="rs-namespace-${RANDOM_SUFFIX}"
WORKGROUP_NAME="rs-workgroup-${RANDOM_SUFFIX}"
ROLE_NAME="RedshiftServerlessS3Role-${RANDOM_SUFFIX}"
SECRET_NAME="redshift-serverless-admin-${RANDOM_SUFFIX}"
```

```
DB_NAME="dev"
ADMIN_USERNAME="admin"

# Generate secure password
echo "Generating secure password..."
ADMIN_PASSWORD=$(generate_secure_password)

# Create secret in AWS Secrets Manager
create_secret "$SECRET_NAME" "$ADMIN_USERNAME" "$ADMIN_PASSWORD" "Admin
 credentials for Redshift Serverless namespace $NAMESPACE_NAME"
if [[ $? -ne 0 ]]; then
  echo "ERROR: Failed to create secret in AWS Secrets Manager"
  exit 1
fi

# Track created resources
CREATED_RESOURCES=()

echo "Using the following resource names:"
echo "- Namespace: $NAMESPACE_NAME"
echo "- Workgroup: $WORKGROUP_NAME"
echo "- IAM Role: $ROLE_NAME"
echo "- Secret: $SECRET_NAME"
echo "- Database: $DB_NAME"
echo "- Admin Username: $ADMIN_USERNAME"
echo "- Admin Password: [STORED IN SECRETS MANAGER]"

# Step 1: Create IAM role for S3 access
echo "Creating IAM role for Redshift Serverless S3 access..."

# Create trust policy document
cat > redshift-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "redshift-serverless.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
EOF

# Create S3 access policy document
cat > redshift-s3-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::redshift-downloads",
        "arn:aws:s3:::redshift-downloads/*"
      ]
    }
  ]
}
EOF

# Create IAM role
echo "Creating IAM role $ROLE_NAME..."
ROLE_OUTPUT=$(aws iam create-role --role-name "$ROLE_NAME" --assume-role-policy-
document file://redshift-trust-policy.json 2>&1)
echo "$ROLE_OUTPUT"
check_error "$ROLE_OUTPUT" "aws iam create-role"
aws iam tag-role --role-name "$ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=redshift-serverless
CREATED_RESOURCES+=("IAM Role: $ROLE_NAME")

# Attach S3 policy to the role
echo "Attaching S3 access policy to role $ROLE_NAME..."
POLICY_OUTPUT=$(aws iam put-role-policy --role-name "$ROLE_NAME" --policy-name
 S3Access --policy-document file://redshift-s3-policy.json 2>&1)
echo "$POLICY_OUTPUT"
check_error "$POLICY_OUTPUT" "aws iam put-role-policy"

# Get the role ARN
ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn' --output
 text)
echo "Role ARN: $ROLE_ARN"
```

```
# Step 2: Create a namespace
echo "Creating Redshift Serverless namespace $NAMESPACE_NAME..."
NAMESPACE_OUTPUT=$(aws redshift-serverless create-namespace \
  --namespace-name "$NAMESPACE_NAME" \
  --admin-username "$ADMIN_USERNAME" \
  --admin-user-password "$ADMIN_PASSWORD" \
  --db-name "$DB_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=redshift-serverless 2>&1)
echo "$NAMESPACE_OUTPUT"
check_error "$NAMESPACE_OUTPUT" "aws redshift-serverless create-namespace"
CREATED_RESOURCES+=("Redshift Serverless Namespace: $NAMESPACE_NAME")

# Wait for namespace to be available
wait_for_resource "namespace" "$NAMESPACE_NAME" 10 30 "aws redshift-serverless
 get-namespace --namespace-name $NAMESPACE_NAME"

# Associate IAM role with namespace
echo "Associating IAM role with namespace..."
UPDATE_NAMESPACE_OUTPUT=$(aws redshift-serverless update-namespace \
  --namespace-name "$NAMESPACE_NAME" \
  --iam-roles "$ROLE_ARN" 2>&1)
echo "$UPDATE_NAMESPACE_OUTPUT"
check_error "$UPDATE_NAMESPACE_OUTPUT" "aws redshift-serverless update-namespace"

# Step 3: Create a workgroup
echo "Creating Redshift Serverless workgroup $WORKGROUP_NAME..."
WORKGROUP_OUTPUT=$(aws redshift-serverless create-workgroup \
  --workgroup-name "$WORKGROUP_NAME" \
  --namespace-name "$NAMESPACE_NAME" \
  --base-capacity 8 \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=redshift-serverless 2>&1)
echo "$WORKGROUP_OUTPUT"
check_error "$WORKGROUP_OUTPUT" "aws redshift-serverless create-workgroup"
CREATED_RESOURCES+=("Redshift Serverless Workgroup: $WORKGROUP_NAME")

# Wait for workgroup to be available
wait_for_resource "workgroup" "$WORKGROUP_NAME" 20 30 "aws redshift-serverless
 get-workgroup --workgroup-name $WORKGROUP_NAME"

# Get workgroup endpoint
WORKGROUP_ENDPOINT=$(aws redshift-serverless get-workgroup \
  --workgroup-name "$WORKGROUP_NAME" \
  --query 'workgroup.endpoint.address' \
  --output text)
```

```
echo "Workgroup endpoint: $WORKGROUP_ENDPOINT"

# Wait additional time for the endpoint to be fully operational
echo "Waiting for endpoint to be fully operational..."
sleep 60

# Step 4: Create tables for sample data
echo "Creating tables for sample data..."

# Create users table
echo "Creating users table..."
USERS_TABLE_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "CREATE TABLE users(
    userid INTEGER NOT NULL DISTKEY SORTKEY,
    username CHAR(8),
    firstname VARCHAR(30),
    lastname VARCHAR(30),
    city VARCHAR(30),
    state CHAR(2),
    email VARCHAR(100),
    phone CHAR(14),
    likesports BOOLEAN,
    liketheatre BOOLEAN,
    likeconcerts BOOLEAN,
    likejazz BOOLEAN,
    likeclassical BOOLEAN,
    likeopera BOOLEAN,
    likerock BOOLEAN,
    likevegas BOOLEAN,
    likebroadway BOOLEAN,
    likemusicals BOOLEAN
  );" 2>&1)
echo "$USERS_TABLE_OUTPUT"
check_error "$USERS_TABLE_OUTPUT" "aws redshift-data execute-statement (users
 table)"
USERS_QUERY_ID=$(echo "$USERS_TABLE_OUTPUT" | grep -o '"Id": "[^"]*' | cut -d'"'
 -f4)

# Wait for query to complete
echo "Waiting for users table creation to complete..."
sleep 5
```

```
# Create event table
echo "Creating event table..."
EVENT_TABLE_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "CREATE TABLE event(
    eventid INTEGER NOT NULL DISTKEY,
    venueid SMALLINT NOT NULL,
    catid SMALLINT NOT NULL,
    dateid SMALLINT NOT NULL SORTKEY,
    eventname VARCHAR(200),
    starttime TIMESTAMP
  );" 2>&1)
echo "$EVENT_TABLE_OUTPUT"
check_error "$EVENT_TABLE_OUTPUT" "aws redshift-data execute-statement (event
 table)"
EVENT_QUERY_ID=$(echo "$EVENT_TABLE_OUTPUT" | grep -o '"Id": "[^"]*' | cut -d'"'
 -f4)

# Wait for query to complete
echo "Waiting for event table creation to complete..."
sleep 5

# Create sales table
echo "Creating sales table..."
SALES_TABLE_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "CREATE TABLE sales(
    salesid INTEGER NOT NULL,
    listid INTEGER NOT NULL DISTKEY,
    sellerid INTEGER NOT NULL,
    buyerid INTEGER NOT NULL,
    eventid INTEGER NOT NULL,
    dateid SMALLINT NOT NULL SORTKEY,
    qtysold SMALLINT NOT NULL,
    pricepaid DECIMAL(8,2),
    commission DECIMAL(8,2),
    saletime TIMESTAMP
  );" 2>&1)
echo "$SALES_TABLE_OUTPUT"
check_error "$SALES_TABLE_OUTPUT" "aws redshift-data execute-statement (sales
 table)"
```

```
SALES_QUERY_ID=$(echo "$SALES_TABLE_OUTPUT" | grep -o '"Id": "[^"]*' | cut -d'"'
 -f4)

# Wait for tables to be created
echo "Waiting for tables to be created..."
sleep 10

# Step 5: Load sample data from Amazon S3
echo "Loading sample data from Amazon S3..."

# Load data into users table
echo "Loading data into users table..."
USERS_LOAD_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "COPY users
    FROM 's3://redshift-downloads/tickit/allusers_pipe.txt'
    DELIMITER '|'
    TIMEFORMAT 'YYYY-MM-DD HH:MI:SS'
    IGNOREHEADER 1
    IAM_ROLE '$ROLE_ARN';" 2>&1)
echo "$USERS_LOAD_OUTPUT"
check_error "$USERS_LOAD_OUTPUT" "aws redshift-data execute-statement (load
 users)"
USERS_LOAD_QUERY_ID=$(echo "$USERS_LOAD_OUTPUT" | grep -o '"Id": "[^"]*' | cut -
d'"' -f4)

# Wait for data loading to complete
echo "Waiting for users data loading to complete..."
sleep 10

# Load data into event table
echo "Loading data into event table..."
EVENT_LOAD_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "COPY event
    FROM 's3://redshift-downloads/tickit/allevents_pipe.txt'
    DELIMITER '|'
    TIMEFORMAT 'YYYY-MM-DD HH:MI:SS'
    IGNOREHEADER 1
    IAM_ROLE '$ROLE_ARN';" 2>&1)
echo "$EVENT_LOAD_OUTPUT"
```

```
check_error "$EVENT_LOAD_OUTPUT" "aws redshift-data execute-statement (load
 event)"
EVENT_LOAD_QUERY_ID=$(echo "$EVENT_LOAD_OUTPUT" | grep -o '"Id": "[^"]*' | cut -
d'"' -f4)

# Wait for data loading to complete
echo "Waiting for event data loading to complete..."
sleep 10

# Load data into sales table
echo "Loading data into sales table..."
SALES_LOAD_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "COPY sales
    FROM 's3://redshift-downloads/tickit/sales_tab.txt'
    DELIMITER '\t'
    TIMEFORMAT 'MM/DD/YYYY HH:MI:SS'
    IGNOREHEADER 1
    IAM_ROLE '$ROLE_ARN';" 2>&1)
echo "$SALES_LOAD_OUTPUT"
check_error "$SALES_LOAD_OUTPUT" "aws redshift-data execute-statement (load
 sales)"
SALES_LOAD_QUERY_ID=$(echo "$SALES_LOAD_OUTPUT" | grep -o '"Id": "[^"]*' | cut -
d'"' -f4)

# Wait for data loading to complete
echo "Waiting for sales data loading to complete..."
sleep 30

# Step 6: Run sample queries
echo "Running sample queries..."

# Query 1: Find top 10 buyers by quantity
echo "Running query: Find top 10 buyers by quantity..."
QUERY1_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "SELECT firstname, lastname, total_quantity
    FROM (SELECT buyerid, sum(qtysold) total_quantity
          FROM sales
          GROUP BY buyerid
          ORDER BY total_quantity desc limit 10) Q, users
    WHERE Q.buyerid = userid
```

```
    ORDER BY Q.total_quantity desc;" 2>&1)
echo "$QUERY1_OUTPUT"
check_error "$QUERY1_OUTPUT" "aws redshift-data execute-statement (query 1)"
QUERY1_ID=$(echo "$QUERY1_OUTPUT" | grep -o '"Id": "[^"]*' | cut -d'"' -f4)

# Wait for query to complete
echo "Waiting for query 1 to complete..."
sleep 10

# Get query 1 results
echo "Getting results for query 1..."
QUERY1_STATUS_OUTPUT=$(aws redshift-data describe-statement --id "$QUERY1_ID"
 2>&1)
echo "$QUERY1_STATUS_OUTPUT"
check_error "$QUERY1_STATUS_OUTPUT" "aws redshift-data describe-statement (query
 1)"

QUERY1_STATUS=$(echo "$QUERY1_STATUS_OUTPUT" | grep -o '"Status": "[^"]*' | cut -
d'"' -f4)
if [ "$QUERY1_STATUS" == "FINISHED" ]; then
  QUERY1_RESULTS=$(aws redshift-data get-statement-result --id "$QUERY1_ID" 2>&1)
  echo "Query 1 Results:"
  echo "$QUERY1_RESULTS"
else
  echo "Query 1 is not yet complete. Status: $QUERY1_STATUS"
  echo "Waiting additional time for query to complete..."
  sleep 20

  # Check again
  QUERY1_STATUS_OUTPUT=$(aws redshift-data describe-statement --id "$QUERY1_ID"
 2>&1)
  QUERY1_STATUS=$(echo "$QUERY1_STATUS_OUTPUT" | grep -o '"Status": "[^"]*' | cut
 -d'"' -f4)

  if [ "$QUERY1_STATUS" == "FINISHED" ]; then
    QUERY1_RESULTS=$(aws redshift-data get-statement-result --id "$QUERY1_ID"
 2>&1)
    echo "Query 1 Results:"
    echo "$QUERY1_RESULTS"
  else
    echo "Query 1 is still not complete. Status: $QUERY1_STATUS"
  fi
fi
```

```
# Query 2: Find events in the 99.9 percentile in terms of all time total sales
echo "Running query: Find events in the 99.9 percentile in terms of all time
 total sales..."
QUERY2_OUTPUT=$(aws redshift-data execute-statement \
  --database "$DB_NAME" \
  --workgroup-name "$WORKGROUP_NAME" \
  --sql "SELECT eventname, total_price
    FROM (SELECT eventid, total_price, ntile(1000) over(order by total_price
 desc) as percentile
          FROM (SELECT eventid, sum(pricepaid) total_price
                FROM sales
                GROUP BY eventid)) Q, event E
    WHERE Q.eventid = E.eventid
    AND percentile = 1
    ORDER BY total_price desc;" 2>&1)
echo "$QUERY2_OUTPUT"
check_error "$QUERY2_OUTPUT" "aws redshift-data execute-statement (query 2)"
QUERY2_ID=$(echo "$QUERY2_OUTPUT" | grep -o '"Id": "[^"]*' | cut -d'"' -f4)

# Wait for query to complete
echo "Waiting for query 2 to complete..."
sleep 10

# Get query 2 results
echo "Getting results for query 2..."
QUERY2_STATUS_OUTPUT=$(aws redshift-data describe-statement --id "$QUERY2_ID"
 2>&1)
echo "$QUERY2_STATUS_OUTPUT"
check_error "$QUERY2_STATUS_OUTPUT" "aws redshift-data describe-statement (query
 2)"

QUERY2_STATUS=$(echo "$QUERY2_STATUS_OUTPUT" | grep -o '"Status": "[^"]*' | cut -
d'"' -f4)
if [ "$QUERY2_STATUS" == "FINISHED" ]; then
  QUERY2_RESULTS=$(aws redshift-data get-statement-result --id "$QUERY2_ID" 2>&1)
  echo "Query 2 Results:"
  echo "$QUERY2_RESULTS"
else
  echo "Query 2 is not yet complete. Status: $QUERY2_STATUS"
  echo "Waiting additional time for query to complete..."
  sleep 20

  # Check again
```

```
  QUERY2_STATUS_OUTPUT=$(aws redshift-data describe-statement --id "$QUERY2_ID"
 2>&1)
  QUERY2_STATUS=$(echo "$QUERY2_STATUS_OUTPUT" | grep -o '"Status": "[^"]*' | cut
 -d'"' -f4)

  if [ "$QUERY2_STATUS" == "FINISHED" ]; then
    QUERY2_RESULTS=$(aws redshift-data get-statement-result --id "$QUERY2_ID"
 2>&1)
    echo "Query 2 Results:"
    echo "$QUERY2_RESULTS"
  else
    echo "Query 2 is still not complete. Status: $QUERY2_STATUS"
  fi
fi

# Summary
echo ""
echo "================================================"
echo "TUTORIAL SUMMARY"
echo "================================================"
echo "You have successfully:"
echo "1. Created a Redshift Serverless namespace and workgroup"
echo "2. Created an IAM role with S3 access permissions"
echo "3. Stored admin credentials securely in AWS Secrets Manager"
echo "4. Created tables for sample data"
echo "5. Loaded sample data from Amazon S3"
echo "6. Run sample queries on the data"
echo ""
echo "Redshift Serverless Resources:"
echo "- Namespace: $NAMESPACE_NAME"
echo "- Workgroup: $WORKGROUP_NAME"
echo "- Database: $DB_NAME"
echo "- Endpoint: $WORKGROUP_ENDPOINT"
echo "- Credentials Secret: $SECRET_NAME"
echo ""
echo "To connect to your Redshift Serverless database using SQL tools:"
echo "- Host: $WORKGROUP_ENDPOINT"
echo "- Database: $DB_NAME"
echo "- Username: $ADMIN_USERNAME"
echo "- Password: Retrieve from AWS Secrets Manager secret '$SECRET_NAME'"
echo ""
echo "To retrieve the password from Secrets Manager (without jq):"
echo "aws secretsmanager get-secret-value --secret-id $SECRET_NAME --query
 'SecretString' --output text | sed -n 's/.*\"password\":\"\([^\"]*\)\".*/\1/p'"
```

```
echo ""

# Clean up temporary files
rm -f redshift-trust-policy.json redshift-s3-policy.json

# Clean up resources
cleanup_resources

echo "Tutorial completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - CreateNamespace
  - CreateRole
  - CreateSecret
  - CreateWorkgroup
  - DeleteNamespace
  - DeleteRole
  - DeleteRolePolicy
  - DeleteSecret
  - DeleteWorkgroup
  - GetNamespace
  - GetRole
  - GetSecretValue
  - GetWorkgroup
  - PutRolePolicy
  - UpdateNamespace

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with configuration management

The following code example shows how to:

- Create an Amazon S3 bucket
- Create an Amazon SNS topic
- Create an IAM role for Config
- Set up the Config configuration recorder
- Set up the Config delivery channel
- Start the configuration recorder
- Verify the Config setup

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS Config Setup Script (v2)
# This script sets up AWS Config with the AWS CLI

# Error handling
set -e
LOGFILE="aws-config-setup-v2.log"
touch $LOGFILE
exec > >(tee -a $LOGFILE)
exec 2>&1

# Function to handle errors
handle_error() {
    echo "ERROR: An error occurred at line $1"
    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Set trap for error handling
```

```
trap 'handle_error $LINENO' ERR

# Function to generate random identifier
generate_random_id() {
    echo $(openssl rand -hex 6)
}

# Function to check if command was successful
check_command() {
    if echo "$1" | grep -i "error" > /dev/null; then
        echo "ERROR: $1"
        return 1
    fi
    return 0
}

# Function to clean up resources
cleanup_resources() {
    if [ -n "$CONFIG_RECORDER_NAME" ]; then
        echo "Stopping configuration recorder..."
        aws configservice stop-configuration-recorder --configuration-recorder-
name "$CONFIG_RECORDER_NAME" 2>/dev/null || true
    fi

    # Check if we created a new delivery channel before trying to delete it
    if [ -n "$DELIVERY_CHANNEL_NAME" ] && [ "$CREATED_NEW_DELIVERY_CHANNEL" =
 "true" ]; then
        echo "Deleting delivery channel..."
        aws configservice delete-delivery-channel --delivery-channel-name
 "$DELIVERY_CHANNEL_NAME" 2>/dev/null || true
    fi

    if [ -n "$CONFIG_RECORDER_NAME" ] && [ "$CREATED_NEW_CONFIG_RECORDER" =
 "true" ]; then
        echo "Deleting configuration recorder..."
        aws configservice delete-configuration-recorder --configuration-recorder-
name "$CONFIG_RECORDER_NAME" 2>/dev/null || true
    fi

    if [ -n "$ROLE_NAME" ]; then
        if [ -n "$POLICY_NAME" ]; then
            echo "Detaching custom policy from role..."
            aws iam delete-role-policy --role-name "$ROLE_NAME" --policy-name
 "$POLICY_NAME" 2>/dev/null || true
```

```
        fi

        if [ -n "$MANAGED_POLICY_ARN" ]; then
            echo "Detaching managed policy from role..."
            aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$MANAGED_POLICY_ARN" 2>/dev/null || true
        fi

        echo "Deleting IAM role..."
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi

    if [ -n "$SNS_TOPIC_ARN" ]; then
        echo "Deleting SNS topic..."
        aws sns delete-topic --topic-arn "$SNS_TOPIC_ARN" 2>/dev/null || true
    fi

    if [ -n "$S3_BUCKET_NAME" ]; then
        echo "Emptying S3 bucket..."
        aws s3 rm "s3://$S3_BUCKET_NAME" --recursive 2>/dev/null || true

        echo "Deleting S3 bucket..."
        if [ "$BUCKET_IS_SHARED" = "false" ]; then
            aws s3api delete-bucket --bucket "$S3_BUCKET_NAME" 2>/dev/null ||
 true
        fi
    fi
}

# Function to display created resources
display_resources() {
    echo ""
    echo "========================================="
    echo "CREATED RESOURCES"
    echo "========================================="
    echo "S3 Bucket: $S3_BUCKET_NAME"
    echo "SNS Topic ARN: $SNS_TOPIC_ARN"
    echo "IAM Role: $ROLE_NAME"
    if [ "$CREATED_NEW_CONFIG_RECORDER" = "true" ]; then
        echo "Configuration Recorder: $CONFIG_RECORDER_NAME (newly created)"
    else
        echo "Configuration Recorder: $CONFIG_RECORDER_NAME (existing)"
    fi
    if [ "$CREATED_NEW_DELIVERY_CHANNEL" = "true" ]; then
```

```
        echo "Delivery Channel: $DELIVERY_CHANNEL_NAME (newly created)"
    else
        echo "Delivery Channel: $DELIVERY_CHANNEL_NAME (existing)"
    fi
    echo "============================================="
}

# Get AWS account ID
echo "Getting AWS account ID..."
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
if [ -z "$ACCOUNT_ID" ]; then
    echo "ERROR: Failed to get AWS account ID"
    exit 1
fi
echo "AWS Account ID: $ACCOUNT_ID"

# Generate random identifier for resources
RANDOM_ID=$(generate_random_id)
echo "Generated random identifier: $RANDOM_ID"

# Step 1: Create an S3 bucket
# Check for shared prereq bucket
PREREQ_BUCKET=$(aws cloudformation describe-stacks --stack-name tutorial-prereqs-
bucket \
    --query 'Stacks[0].Outputs[?OutputKey==`BucketName`].OutputValue' --output
 text 2>/dev/null)
if [ -n "$PREREQ_BUCKET" ] && [ "$PREREQ_BUCKET" != "None" ]; then
    S3_BUCKET_NAME="$PREREQ_BUCKET"
    BUCKET_IS_SHARED=true
    echo "Using shared bucket: $S3_BUCKET_NAME"
else
    BUCKET_IS_SHARED=false
    S3_BUCKET_NAME="configservice-${RANDOM_ID}"
    echo "Creating S3 bucket: $S3_BUCKET_NAME"
fi

# Get the current region
AWS_REGION=$(aws configure get region)
if [ -z "$AWS_REGION" ]; then
    AWS_REGION="us-east-1"  # Default to us-east-1 if no region is configured
fi
echo "Using AWS Region: $AWS_REGION"

# Create bucket with appropriate command based on region
```

```
if [ "$BUCKET_IS_SHARED" = "false" ]; then
    if [ "$AWS_REGION" = "us-east-1" ]; then
        BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME")
    else
        BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME" --
create-bucket-configuration LocationConstraint="$AWS_REGION")
    fi
    check_command "$BUCKET_RESULT"
    echo "S3 bucket created: $S3_BUCKET_NAME"

    aws s3api put-bucket-tagging --bucket "$S3_BUCKET_NAME" --tagging
 'TagSet=[{Key=project,Value=doc-smith},{Key=tutorial,Value=aws-config-gs}]'
    echo "Tags applied to S3 bucket"
else
    echo "Using shared bucket: $S3_BUCKET_NAME (skipping creation)"
fi

# Block public access for the bucket
aws s3api put-public-access-block \
    --bucket "$S3_BUCKET_NAME" \
    --public-access-block-configuration
 "BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets
echo "Public access blocked for bucket"

# Step 2: Create an SNS topic
TOPIC_NAME="config-topic-${RANDOM_ID}"
echo "Creating SNS topic: $TOPIC_NAME"
SNS_RESULT=$(aws sns create-topic --name "$TOPIC_NAME" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-config-gs)
check_command "$SNS_RESULT"
SNS_TOPIC_ARN=$(echo "$SNS_RESULT" | grep -o 'arn:aws:sns:[^"]*')
echo "SNS topic created: $SNS_TOPIC_ARN"

# Step 3: Create an IAM role for AWS Config
ROLE_NAME="config-role-${RANDOM_ID}"
POLICY_NAME="config-delivery-permissions"
MANAGED_POLICY_ARN="arn:aws:iam::aws:policy/service-role/AWS_ConfigRole"

echo "Creating trust policy document..."
cat > config-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "config.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

echo "Creating IAM role: $ROLE_NAME"
ROLE_RESULT=$(aws iam create-role --role-name "$ROLE_NAME" --assume-role-policy-
document file://config-trust-policy.json)
check_command "$ROLE_RESULT"
ROLE_ARN=$(echo "$ROLE_RESULT" | grep -o 'arn:aws:iam::[^"]*' | head -1)
echo "IAM role created: $ROLE_ARN"

aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-config-gs
echo "Tags applied to IAM role"

echo "Attaching AWS managed policy to role..."
ATTACH_RESULT=$(aws iam attach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$MANAGED_POLICY_ARN")
check_command "$ATTACH_RESULT"
echo "AWS managed policy attached"

echo "Creating custom policy document for S3 and SNS access..."
cat > config-delivery-permissions.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::${S3_BUCKET_NAME}/AWSLogs/${ACCOUNT_ID}/*",
      "Condition": {
        "StringLike": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        }
      }
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketAcl"
      ],
      "Resource": "arn:aws:s3:::${S3_BUCKET_NAME}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": "${SNS_TOPIC_ARN}"
    }
  ]
}
EOF

echo "Attaching custom policy to role..."
POLICY_RESULT=$(aws iam put-role-policy --role-name "$ROLE_NAME" --policy-name
 "$POLICY_NAME" --policy-document file://config-delivery-permissions.json)
check_command "$POLICY_RESULT"
echo "Custom policy attached"

# Wait for IAM role to propagate
echo "Waiting for IAM role to propagate (15 seconds)..."
sleep 15

# Step 4: Check if configuration recorder already exists
CONFIG_RECORDER_NAME="default"
CREATED_NEW_CONFIG_RECORDER="false"

echo "Checking for existing configuration recorder..."
EXISTING_RECORDERS=$(aws configservice describe-configuration-recorders 2>/dev/
null || echo "")
if echo "$EXISTING_RECORDERS" | grep -q "name"; then
    echo "Configuration recorder already exists. Will update it."
    # Get the name of the existing recorder
    CONFIG_RECORDER_NAME=$(echo "$EXISTING_RECORDERS" | grep -o '"name": "[^"]*"'
 | head -1 | cut -d'"' -f4)
    echo "Using existing configuration recorder: $CONFIG_RECORDER_NAME"
else
    echo "No existing configuration recorder found. Will create a new one."
    CREATED_NEW_CONFIG_RECORDER="true"
```

```
fi

echo "Creating configuration recorder configuration..."
cat > configurationRecorder.json << EOF
{
  "name": "${CONFIG_RECORDER_NAME}",
  "roleARN": "${ROLE_ARN}",
  "recordingMode": {
    "recordingFrequency": "CONTINUOUS"
  }
}
EOF

echo "Creating recording group configuration..."
cat > recordingGroup.json << EOF
{
  "allSupported": true,
  "includeGlobalResourceTypes": true
}
EOF

echo "Setting up configuration recorder..."
RECORDER_RESULT=$(aws configservice put-configuration-recorder --configuration-
recorder file://configurationRecorder.json --recording-group file://
recordingGroup.json)
check_command "$RECORDER_RESULT"
echo "Configuration recorder set up"

if [ "$CREATED_NEW_CONFIG_RECORDER" = "true" ]; then
    aws configservice tag-resource --resource-arn "arn:aws:config:
${AWS_REGION}:${ACCOUNT_ID}:config-recorder/${CONFIG_RECORDER_NAME}" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-config-gs
    echo "Tags applied to configuration recorder"
fi

# Step 5: Check if delivery channel already exists
DELIVERY_CHANNEL_NAME="default"
CREATED_NEW_DELIVERY_CHANNEL="false"

echo "Checking for existing delivery channel..."
EXISTING_CHANNELS=$(aws configservice describe-delivery-channels 2>/dev/null ||
 echo "")
if echo "$EXISTING_CHANNELS" | grep -q "name"; then
    echo "Delivery channel already exists."
```

```
    # Get the name of the existing channel
    DELIVERY_CHANNEL_NAME=$(echo "$EXISTING_CHANNELS" | grep -o '"name": "[^"]*"'
 | head -1 | cut -d'"' -f4)
    echo "Using existing delivery channel: $DELIVERY_CHANNEL_NAME"

    # Update the existing delivery channel
    echo "Creating delivery channel configuration for update..."
    cat > deliveryChannel.json << EOF
{
  "name": "${DELIVERY_CHANNEL_NAME}",
  "s3BucketName": "${S3_BUCKET_NAME}",
  "snsTopicARN": "${SNS_TOPIC_ARN}",
  "configSnapshotDeliveryProperties": {
    "deliveryFrequency": "Six_Hours"
  }
}
EOF

    echo "Updating delivery channel..."
    CHANNEL_RESULT=$(aws configservice put-delivery-channel --delivery-channel
 file://deliveryChannel.json)
    check_command "$CHANNEL_RESULT"
    echo "Delivery channel updated"
else
    echo "No existing delivery channel found. Will create a new one."
    CREATED_NEW_DELIVERY_CHANNEL="true"

    echo "Creating delivery channel configuration..."
    cat > deliveryChannel.json << EOF
{
  "name": "${DELIVERY_CHANNEL_NAME}",
  "s3BucketName": "${S3_BUCKET_NAME}",
  "snsTopicARN": "${SNS_TOPIC_ARN}",
  "configSnapshotDeliveryProperties": {
    "deliveryFrequency": "Six_Hours"
  }
}
EOF

    echo "Creating delivery channel..."
    CHANNEL_RESULT=$(aws configservice put-delivery-channel --delivery-channel
 file://deliveryChannel.json)
    check_command "$CHANNEL_RESULT"
    echo "Delivery channel created"
```

```
    aws configservice tag-resource --resource-arn "arn:aws:config:
${AWS_REGION}:${ACCOUNT_ID}:delivery-channel/${DELIVERY_CHANNEL_NAME}" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-config-gs
    echo "Tags applied to delivery channel"
fi

# Step 6: Start the configuration recorder
echo "Checking configuration recorder status..."
RECORDER_STATUS=$(aws configservice describe-configuration-recorder-status 2>/
dev/null || echo "")
if echo "$RECORDER_STATUS" | grep -q '"recording": true'; then
    echo "Configuration recorder is already running."
else
    echo "Starting configuration recorder..."
    START_RESULT=$(aws configservice start-configuration-recorder --
configuration-recorder-name "$CONFIG_RECORDER_NAME")
    check_command "$START_RESULT"
    echo "Configuration recorder started"
fi

# Step 7: Verify the AWS Config setup
echo "Verifying delivery channel..."
VERIFY_CHANNEL=$(aws configservice describe-delivery-channels)
check_command "$VERIFY_CHANNEL"
echo "$VERIFY_CHANNEL"

echo "Verifying configuration recorder..."
VERIFY_RECORDER=$(aws configservice describe-configuration-recorders)
check_command "$VERIFY_RECORDER"
echo "$VERIFY_RECORDER"

echo "Verifying configuration recorder status..."
VERIFY_STATUS=$(aws configservice describe-configuration-recorder-status)
check_command "$VERIFY_STATUS"
echo "$VERIFY_STATUS"

# Display created resources
display_resources

# Ask if user wants to clean up resources
echo ""
echo "============================================"
echo "CLEANUP CONFIRMATION"
```

```
echo "========================================="
echo "Do you want to clean up all created resources? (y/n): "
CLEANUP_CHOICE='y'

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    echo "Cleaning up resources..."
    cleanup_resources
    echo "Cleanup completed."
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later."
fi

echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateBucket
  - CreateRole
  - CreateTopic
  - DeleteBucket
  - DeleteConfigurationRecorder
  - DeleteDeliveryChannel
  - DeleteRole
  - DeleteRolePolicy
  - DeleteTopic
  - DescribeConfigurationRecorderStatus
  - DescribeConfigurationRecorders
  - DescribeDeliveryChannels
  - DetachRolePolicy
  - GetCallerIdentity
  - PutConfigurationRecorder
  - PutDeliveryChannel
  - PutPublicAccessBlock
  - PutRolePolicy

- Rm
- StartConfigurationRecorder
- StopConfigurationRecorder

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with internet of things device protection

The following code example shows how to:

- Create Required IAM Roles
- Enable IoT Device Defender Audit Checks
- Run an On-Demand Audit
- Create a Mitigation Action
- Apply Mitigation Actions to Findings
- Set Up SNS Notifications (Optional)
- Enable IoT Logging

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS IoT Device Defender Getting Started Script
# This script demonstrates how to use AWS IoT Device Defender to enable audit
 checks,
# view audit results, create mitigation actions, and apply them to findings.
```

```
set -euo pipefail

# Set up logging
LOG_FILE="iot-device-defender-script-$(date +%Y%m%d%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "================================================="
echo "AWS IoT Device Defender Getting Started Script"
echo "================================================="
echo "Starting script execution at $(date)"
echo ""

# Function to check for errors in command output
check_error() {
    if echo "$1" | grep -iE "An error occurred|Exception|Failed|usage: aws" > /
dev/null; then
        echo "ERROR: Command failed with the following output:"
        echo "$1"
        return 1
    fi
    return 0
}

# Function to safely extract JSON values using jq
extract_json_value() {
    local json="$1"
    local key="$2"
    echo "$json" | jq -r ".${key} // empty" 2>/dev/null || echo ""
}

# Function to validate JSON
validate_json() {
    local json="$1"
    echo "$json" | jq empty 2>/dev/null
}

# Function to check AWS CLI availability
check_aws_cli() {
    if ! command -v aws &> /dev/null; then
        echo "ERROR: AWS CLI is not installed or not in PATH"
        return 1
    fi
    if ! command -v jq &> /dev/null; then
        echo "ERROR: jq is not installed or not in PATH"
```

```
        return 1
    fi
    return 0
}

# Function to get AWS account ID
get_account_id() {
    local account_id
    account_id=$(aws sts get-caller-identity --query 'Account' --output text 2>/
dev/null) || true
    if [ -z "$account_id" ]; then
        echo "ERROR: Could not retrieve AWS account ID"
        return 1
    fi
    echo "$account_id"
    return 0
}

# Function to create IAM roles with retry logic
create_iam_role() {
    local ROLE_NAME=$1
    local TRUST_POLICY=$2
    local MANAGED_POLICY=$3
    local RETRY_COUNT=0
    local MAX_RETRIES=3

    echo "Creating IAM role: $ROLE_NAME"

    # Validate trust policy JSON
    if ! validate_json "$TRUST_POLICY"; then
        echo "ERROR: Invalid trust policy JSON for role $ROLE_NAME"
        return 1
    fi

    # Check if role already exists
    if aws iam get-role --role-name "$ROLE_NAME" >/dev/null 2>&1; then
        echo "Role $ROLE_NAME already exists, skipping creation"
        ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn'
 --output text 2>/dev/null) || true
        if [ -z "$ROLE_ARN" ]; then
            echo "ERROR: Could not retrieve ARN for existing role $ROLE_NAME"
            return 1
        fi
        echo "Role ARN: $ROLE_ARN"
```

```
        return 0
    fi

    # Create the role with trust policy and retry logic
    while [ $RETRY_COUNT -lt $MAX_RETRIES ]; do
        ROLE_RESULT=$(aws iam create-role \
            --role-name "$ROLE_NAME" \
            --assume-role-policy-document "$TRUST_POLICY" 2>&1) || true

        if check_error "$ROLE_RESULT"; then
            break
        fi

        RETRY_COUNT=$((RETRY_COUNT + 1))
        if [ $RETRY_COUNT -lt $MAX_RETRIES ]; then
            echo "Retrying role creation (attempt $((RETRY_COUNT + 1))/
$MAX_RETRIES)..."
            sleep $((RETRY_COUNT * 2))
        fi
    done

    if ! check_error "$ROLE_RESULT"; then
        echo "Failed to create role $ROLE_NAME after $MAX_RETRIES attempts"
        return 1
    fi

    aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-device-defender-gs 2>&1 || true

    # For IoT logging role, create an inline policy instead of using a managed
 policy
    if [[ "$ROLE_NAME" == "AWSIoTLoggingRole" ]]; then
        local LOGGING_POLICY
        LOGGING_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
```

```
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        }
    ]
}
EOF
)

        if ! validate_json "$LOGGING_POLICY"; then
            echo "ERROR: Invalid logging policy JSON"
            return 1
        fi

        POLICY_RESULT=$(aws iam put-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-name "${ROLE_NAME}Policy" \
            --policy-document "$LOGGING_POLICY" 2>&1) || true

        if ! check_error "$POLICY_RESULT"; then
            echo "Failed to attach inline policy to role $ROLE_NAME"
            return 1
        fi
    elif [[ "$ROLE_NAME" == "IoTMitigationActionErrorLoggingRole" ]]; then
        local MITIGATION_POLICY
        MITIGATION_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:UpdateCACertificate",
                "iot:UpdateCertificate",
                "iot:SetV2LoggingOptions",
                "iot:SetLoggingOptions",
                "iot:AddThingToThingGroup"
            ],
            "Resource": "arn:aws:iot:*:*:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
        }
    ]
}
EOF
)

        if ! validate_json "$MITIGATION_POLICY"; then
            echo "ERROR: Invalid mitigation policy JSON"
            return 1
        fi

        POLICY_RESULT=$(aws iam put-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-name "${ROLE_NAME}Policy" \
            --policy-document "$MITIGATION_POLICY" 2>&1) || true

        if ! check_error "$POLICY_RESULT"; then
            echo "Failed to attach inline policy to role $ROLE_NAME"
            return 1
        fi
    else
        # Attach managed policy to role if provided
        if [ -n "$MANAGED_POLICY" ]; then
            ATTACH_RESULT=$(aws iam attach-role-policy \
                --role-name "$ROLE_NAME" \
                --policy-arn "$MANAGED_POLICY" 2>&1) || true

            if ! check_error "$ATTACH_RESULT"; then
                echo "Failed to attach policy to role $ROLE_NAME"
                return 1
            fi
        fi
    fi

    echo "Role $ROLE_NAME created successfully"

    # Get the role ARN with error handling
```

```
    ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn' --
output text 2>/dev/null) || true
    if [ -z "$ROLE_ARN" ]; then
        echo "ERROR: Could not retrieve ARN for newly created role $ROLE_NAME"
        return 1
    fi
    echo "Role ARN: $ROLE_ARN"
    return 0
}

# Array to store created resources for cleanup
declare -a CREATED_RESOURCES

# Validate prerequisites
echo "Validating prerequisites..."
if ! check_aws_cli; then
    echo "ERROR: Prerequisites not met"
    exit 1
fi

ACCOUNT_ID=$(get_account_id) || exit 1
echo "AWS Account ID: $ACCOUNT_ID"
echo ""

# Step 1: Create IAM roles needed for the tutorial
echo "=================================================="
echo "Step 1: Creating required IAM roles"
echo "=================================================="

# Create IoT Device Defender Audit role
IOT_DEFENDER_AUDIT_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

```
)

if ! create_iam_role "AWSIoTDeviceDefenderAuditRole"
 "$IOT_DEFENDER_AUDIT_TRUST_POLICY" "arn:aws:iam::aws:policy/service-role/
AWSIoTDeviceDefenderAudit"; then
    echo "ERROR: Failed to create audit role"
    exit 1
fi
AUDIT_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: AWSIoTDeviceDefenderAuditRole")

# Create IoT Logging role
IOT_LOGGING_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)

if ! create_iam_role "AWSIoTLoggingRole" "$IOT_LOGGING_TRUST_POLICY" ""; then
    echo "ERROR: Failed to create logging role"
    exit 1
fi
LOGGING_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: AWSIoTLoggingRole")

# Create IoT Mitigation Action role
IOT_MITIGATION_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)

if ! create_iam_role "IoTMitigationActionErrorLoggingRole"
 "$IOT_MITIGATION_TRUST_POLICY" ""; then
    echo "ERROR: Failed to create mitigation role"
    exit 1
fi
MITIGATION_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: IoTMitigationActionErrorLoggingRole")

# Wait for IAM role propagation
echo "Waiting for IAM role propagation..."
sleep 5

# Step 2: Enable audit checks
echo ""
echo "================================================="
echo "Step 2: Enabling AWS IoT Device Defender audit checks"
echo "================================================="

# Get current audit configuration
echo "Getting current audit configuration..."
CURRENT_CONFIG=$(aws iot describe-account-audit-configuration --output json 2>&1)
 || true
if validate_json "$CURRENT_CONFIG"; then
    echo "$CURRENT_CONFIG" | jq '.' 2>/dev/null || echo "Could not parse current
 configuration"
fi

# Enable specific audit checks with proper JSON escaping
echo "Enabling audit checks..."
AUDIT_CONFIG='{"LOGGING_DISABLED_CHECK":{"enabled":true}}'

if ! validate_json "$AUDIT_CONFIG"; then
    echo "ERROR: Invalid audit configuration JSON"
    exit 1
fi
```

```
UPDATE_RESULT=$(aws iot update-account-audit-configuration \
  --role-arn "$AUDIT_ROLE_ARN" \
  --audit-check-configurations "$AUDIT_CONFIG" 2>&1) || true

if ! check_error "$UPDATE_RESULT"; then
    echo "Failed to update audit configuration"
    exit 1
fi

echo "Audit checks enabled successfully"

# Step 3: Run an on-demand audit
echo ""
echo "=================================================="
echo "Step 3: Running an on-demand audit"
echo "=================================================="

echo "Starting on-demand audit task..."
AUDIT_TASK_RESULT=$(aws iot start-on-demand-audit-task \
  --target-check-names LOGGING_DISABLED_CHECK --output json 2>&1) || true

if ! check_error "$AUDIT_TASK_RESULT"; then
    echo "Failed to start on-demand audit task"
    exit 1
fi

TASK_ID=$(extract_json_value "$AUDIT_TASK_RESULT" "taskId")
if [ -z "$TASK_ID" ]; then
    echo "ERROR: Could not extract task ID from response"
    exit 1
fi

echo "Audit task started with ID: $TASK_ID"
CREATED_RESOURCES+=("Audit Task: $TASK_ID")

# Tag the audit task via IoT service
aws iot tag-resource --resource-arn "arn:aws:iot:$(aws configure get region):
${ACCOUNT_ID}:audittask/${TASK_ID}" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-device-defender-gs 2>&1 || true

# Wait for the audit task to complete
echo "Waiting for audit task to complete (this may take a few minutes)..."
TASK_STATUS="IN_PROGRESS"
TIMEOUT=0
```

```
MAX_TIMEOUT=600
POLL_INTERVAL=15

while [ "$TASK_STATUS" != "COMPLETED" ]; do
    if [ $TIMEOUT -ge $MAX_TIMEOUT ]; then
        echo "WARNING: Audit task did not complete within ${MAX_TIMEOUT} seconds,
 continuing..."
        break
    fi

    sleep "$POLL_INTERVAL"
    TIMEOUT=$((TIMEOUT + POLL_INTERVAL))

    TASK_DETAILS=$(aws iot describe-audit-task --task-id "$TASK_ID" --output json
 2>&1) || true
    if validate_json "$TASK_DETAILS"; then
        TASK_STATUS=$(extract_json_value "$TASK_DETAILS" "taskStatus")
        echo "Current task status: $TASK_STATUS (elapsed: ${TIMEOUT}s)"

        if [ "$TASK_STATUS" = "FAILED" ]; then
            echo "WARNING: Audit task failed, continuing with script..."
            FAILURE_REASON=$(extract_json_value "$TASK_DETAILS"
 "taskStatistics.failedChecksNotApplicable")
            if [ -n "$FAILURE_REASON" ]; then
                echo "Reason: $FAILURE_REASON"
            fi
            break
        fi
    else
        echo "WARNING: Could not parse task details, retrying..."
    fi
done

echo "Audit task processing completed"

# Get audit findings (non-blocking)
echo "Getting audit findings..."
FINDINGS=$(aws iot list-audit-findings \
  --task-id "$TASK_ID" --output json 2>&1) || true

if validate_json "$FINDINGS"; then
    FINDING_COUNT=$(echo "$FINDINGS" | jq '.findings | length' 2>/dev/null ||
 echo "0")
    echo "Audit findings count: $FINDING_COUNT"
```

```
    if [ "$FINDING_COUNT" -gt 0 ]; then
        echo "Sample finding:"
        echo "$FINDINGS" | jq '.findings[0]' 2>/dev/null || echo "Could not parse
 finding"
    fi
else
    echo "WARNING: Could not parse audit findings response"
    FINDINGS='{"findings":[]}'
fi

# Check if we have any non-compliant findings
FINDING_ID=$(extract_json_value "$FINDINGS" "findings[0].findingId")
if [ -n "$FINDING_ID" ]; then
    echo "Found non-compliant finding with ID: $FINDING_ID"
    HAS_FINDINGS=true
else
    echo "No non-compliant findings detected"
    HAS_FINDINGS=false
fi

# Step 4: Create a mitigation action
echo ""
echo "================================================="
echo "Step 4: Creating a mitigation action"
echo "================================================="

# Check if mitigation action already exists and delete it
if aws iot describe-mitigation-action --action-name "EnableErrorLoggingAction" >/
dev/null 2>&1; then
    echo "Mitigation action 'EnableErrorLoggingAction' already exists, deleting
 it first..."
    aws iot delete-mitigation-action --action-name "EnableErrorLoggingAction"
 2>&1 || true
    sleep 2
fi

echo "Creating mitigation action to enable AWS IoT logging..."

# Build mitigation action parameters JSON
MITIGATION_PARAMS=$(cat <<EOF
{
  "enableIoTLoggingParams": {
    "roleArnForLogging": "$LOGGING_ROLE_ARN",
    "logLevel": "ERROR"
```

```
  }
}
EOF
)

if ! validate_json "$MITIGATION_PARAMS"; then
    echo "ERROR: Invalid mitigation parameters JSON"
    exit 1
fi

MITIGATION_RESULT=$(aws iot create-mitigation-action \
  --action-name "EnableErrorLoggingAction" \
  --role-arn "$MITIGATION_ROLE_ARN" \
  --action-params "$MITIGATION_PARAMS" --output json 2>&1) || true

if ! check_error "$MITIGATION_RESULT"; then
    echo "Failed to create mitigation action"
    exit 1
fi

if validate_json "$MITIGATION_RESULT"; then
    echo "Mitigation action created successfully"
    MITIGATION_ACTION_ARN=$(extract_json_value "$MITIGATION_RESULT" "actionArn")
    if [ -n "$MITIGATION_ACTION_ARN" ]; then
        echo "Mitigation Action ARN: $MITIGATION_ACTION_ARN"
        aws iot tag-resource --resource-arn "$MITIGATION_ACTION_ARN" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-iot-device-defender-gs 2>&1
 || true
    fi
else
    echo "WARNING: Could not validate mitigation action response, but action may
 have been created"
fi

CREATED_RESOURCES+=("Mitigation Action: EnableErrorLoggingAction")

# Step 5: Apply mitigation action to findings (if any)
if [ "$HAS_FINDINGS" = true ]; then
    echo ""
    echo "================================================="
    echo "Step 5: Applying mitigation action to findings"
    echo "================================================="

    MITIGATION_TASK_ID="MitigationTask-$(date +%s)"
```

```
    echo "Starting mitigation actions task with ID: $MITIGATION_TASK_ID"

    # Build target JSON
    TARGET_JSON=$(cat <<EOF
{
  "findingIds": ["$FINDING_ID"]
}
EOF
)

    if ! validate_json "$TARGET_JSON"; then
        echo "ERROR: Invalid target JSON"
        exit 1
    fi

    # Build audit check to actions mapping JSON
    AUDIT_CHECK_MAPPING=$(cat <<EOF
{
  "LOGGING_DISABLED_CHECK": ["EnableErrorLoggingAction"]
}
EOF
)

    if ! validate_json "$AUDIT_CHECK_MAPPING"; then
        echo "ERROR: Invalid audit check mapping JSON"
        exit 1
    fi

    MITIGATION_TASK_RESULT=$(aws iot start-audit-mitigation-actions-task \
      --task-id "$MITIGATION_TASK_ID" \
      --target "$TARGET_JSON" \
      --audit-check-to-actions-mapping "$AUDIT_CHECK_MAPPING" --output json 2>&1)
 || true

    if ! check_error "$MITIGATION_TASK_RESULT"; then
        echo "WARNING: Failed to start mitigation actions task, continuing..."
    else
        echo "Mitigation actions task started successfully"
        CREATED_RESOURCES+=("Mitigation Task: $MITIGATION_TASK_ID")
    fi
else
    echo ""
    echo "=================================================="
    echo "Step 5: Skipping mitigation action application (no findings)"
```

```
    echo "================================================="
fi

# Step 6: Set up SNS notifications (optional)
echo ""
echo "================================================="
echo "Step 6: Setting up SNS notifications"
echo "================================================="

# Check if SNS topic already exists
SNS_TOPICS=$(aws sns list-topics --output json 2>&1) || true
TOPIC_ARN=""
if validate_json "$SNS_TOPICS"; then
    TOPIC_ARN=$(echo "$SNS_TOPICS" | jq -r '.Topics[] | select(.TopicArn |
 contains("IoTDDNotifications")) | .TopicArn' 2>/dev/null | head -1 || echo "")
fi

if [ -n "$TOPIC_ARN" ]; then
    echo "SNS topic 'IoTDDNotifications' already exists, using existing topic..."
    echo "Topic ARN: $TOPIC_ARN"
else
    echo "Creating SNS topic for notifications..."
    SNS_RESULT=$(aws sns create-topic --name "IoTDDNotifications" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-iot-device-defender-gs --
output json 2>&1) || true

    if ! check_error "$SNS_RESULT"; then
        echo "WARNING: Failed to create SNS topic, continuing..."
        SNS_RESULT=""
    else
        TOPIC_ARN=$(extract_json_value "$SNS_RESULT" "TopicArn")
        if [ -n "$TOPIC_ARN" ]; then
            echo "SNS topic created with ARN: $TOPIC_ARN"
            CREATED_RESOURCES+=("SNS Topic: IoTDDNotifications")
        fi
    fi
fi

if [ -n "$TOPIC_ARN" ]; then
    echo "Updating audit configuration to enable SNS notifications..."

    # Build SNS notification configuration JSON
    SNS_CONFIG=$(cat <<EOF
{
```

```
  "SNS": {
    "targetArn": "$TOPIC_ARN",
    "roleArn": "$AUDIT_ROLE_ARN",
    "enabled": true
  }
}
EOF
)

    if ! validate_json "$SNS_CONFIG"; then
        echo "ERROR: Invalid SNS configuration JSON"
        exit 1
    fi

    SNS_UPDATE_RESULT=$(aws iot update-account-audit-configuration \
      --audit-notification-target-configurations "$SNS_CONFIG" 2>&1) || true

    if ! check_error "$SNS_UPDATE_RESULT"; then
        echo "WARNING: Failed to update audit configuration for SNS
 notifications"
    else
        echo "SNS notifications enabled successfully"
    fi
else
    echo "Skipping SNS configuration due to topic creation failure"
fi

# Step 7: Enable AWS IoT logging
echo ""
echo "================================================="
echo "Step 7: Enabling AWS IoT logging"
echo "================================================="

echo "Setting up AWS IoT logging options..."

LOGGING_RESULT=$(aws iot set-v2-logging-options \
  --role-arn "$LOGGING_ROLE_ARN" \
  --default-log-level "ERROR" 2>&1) || true

if ! check_error "$LOGGING_RESULT"; then
    echo "V2 logging setup failed, trying v1 logging..."

    V1_LOGGING_CONFIG=$(cat <<EOF
{
```

```
  "roleArn": "$LOGGING_ROLE_ARN",
  "logLevel": "ERROR"
}
EOF
)

    if ! validate_json "$V1_LOGGING_CONFIG"; then
        echo "ERROR: Invalid v1 logging configuration JSON"
        exit 1
    fi

    LOGGING_RESULT_V1=$(aws iot set-logging-options \
      --logging-options-payload "$V1_LOGGING_CONFIG" 2>&1) || true

    if ! check_error "$LOGGING_RESULT_V1"; then
        echo "WARNING: Failed to set up AWS IoT logging with both v1 and v2
 methods, continuing..."
    else
        echo "AWS IoT v1 logging enabled successfully"
    fi
else
    echo "AWS IoT v2 logging enabled successfully"
fi

# Verify logging is enabled
echo "Verifying logging configuration..."
LOGGING_CONFIG=$(aws iot get-v2-logging-options --output json 2>&1) || true
if [ -n "$LOGGING_CONFIG" ] && ! check_error "$LOGGING_CONFIG" && validate_json
 "$LOGGING_CONFIG"; then
    echo "Logging configuration verified"
    echo "$LOGGING_CONFIG" | jq '.' 2>/dev/null || echo "Configuration retrieved
 but could not display details"
else
    echo "Could not verify logging configuration, but setup completed"
fi

# Script completed successfully
echo ""
echo "================================================="
echo "AWS IoT Device Defender setup completed successfully!"
echo "================================================="
echo "The following resources were created:"
for resource in "${CREATED_RESOURCES[@]+"${CREATED_RESOURCES[@]}"}"; do
    echo "- $resource"
```

```
done
echo ""

# Cleanup phase
echo "=========================================="
echo "CLEANUP"
echo "=========================================="
echo "Starting automatic cleanup of resources..."
echo "Waiting 10 seconds before cleanup to allow resource stabilization..."
sleep 10

# Disable AWS IoT logging
echo "Disabling AWS IoT logging..."

DISABLE_V2_RESULT=$(aws iot set-v2-logging-options \
  --default-log-level "DISABLED" 2>&1) || true

if check_error "$DISABLE_V2_RESULT"; then
    echo "V2 logging disabled successfully"
else
    echo "Attempting v1 logging disable..."

    V1_DISABLE_CONFIG=$(cat <<'EOF'
{
  "logLevel": "DISABLED"
}
EOF
)

    DISABLE_V1_RESULT=$(aws iot set-logging-options \
      --logging-options-payload "$V1_DISABLE_CONFIG" 2>&1) || true

    if check_error "$DISABLE_V1_RESULT"; then
        echo "V1 logging disabled successfully"
    else
        echo "WARNING: Could not disable logging"
    fi
fi

# Delete mitigation action
echo "Deleting mitigation action..."
aws iot delete-mitigation-action --action-name "EnableErrorLoggingAction" 2>&1 ||
 true
```

```
# Reset audit configuration
echo "Resetting IoT Device Defender audit configuration..."
RESET_AUDIT_CONFIG='{"LOGGING_DISABLED_CHECK":{"enabled":false}}'
aws iot update-account-audit-configuration \
  --audit-check-configurations "$RESET_AUDIT_CONFIG" 2>&1 || true

# Delete SNS topic
echo "Deleting SNS topic..."
if [ -n "${TOPIC_ARN:-}" ] && [ "$TOPIC_ARN" != "null" ]; then
    aws sns delete-topic --topic-arn "$TOPIC_ARN" 2>&1 || true
fi

# Clean up IAM roles with improved error handling
echo "Cleaning up IAM roles..."

cleanup_role() {
    local role_name=$1
    echo "Cleaning up role: $role_name"

    if aws iam get-role --role-name "$role_name" >/dev/null 2>&1; then
        ROLE_POLICIES=$(aws iam list-role-policies --role-name "$role_name" --
output json 2>&1 || echo '{"PolicyNames":[]}')
        if validate_json "$ROLE_POLICIES"; then
            while IFS= read -r policy_name; do
                if [ -n "$policy_name" ] && [ "$policy_name" != "null" ]; then
                    echo "  Deleting inline policy: $policy_name"
                    aws iam delete-role-policy \
                        --role-name "$role_name" \
                        --policy-name "$policy_name" 2>&1 || true
                fi
            done < <(echo "$ROLE_POLICIES" | jq -r '.PolicyNames[]' 2>/dev/null
 || echo "")
        fi

        ATTACHED_POLICIES=$(aws iam list-attached-role-policies --role-name
 "$role_name" --output json 2>&1 || echo '{"AttachedPolicies":[]}')
        if validate_json "$ATTACHED_POLICIES"; then
            while IFS= read -r policy_arn; do
                if [ -n "$policy_arn" ] && [ "$policy_arn" != "null" ]; then
                    echo "  Detaching managed policy: $policy_arn"
                    aws iam detach-role-policy \
                        --role-name "$role_name" \
                        --policy-arn "$policy_arn" 2>&1 || true
                fi
```

```
            done < <(echo "$ATTACHED_POLICIES" | jq -r
 '.AttachedPolicies[].PolicyArn' 2>/dev/null || echo "")
        fi

        echo "  Deleting role: $role_name"
        aws iam delete-role --role-name "$role_name" 2>&1 || true
    else
        echo "  Role $role_name does not exist or already deleted"
    fi
}

cleanup_role "AWSIoTDeviceDefenderAuditRole"
cleanup_role "AWSIoTLoggingRole"
cleanup_role "IoTMitigationActionErrorLoggingRole"

echo "Cleanup completed successfully"

echo ""
echo "Script execution completed at $(date)"
echo "Log file: $LOG_FILE"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateMitigationAction
  - CreateRole
  - CreateTopic
  - DeleteAccountAuditConfiguration
  - DeleteMitigationAction
  - DeleteRole
  - DeleteRolePolicy
  - DeleteTopic
  - DescribeAccountAuditConfiguration
  - DescribeAuditTask
  - DetachRolePolicy
  - GetLoggingOptions
  - GetRole
  - GetV2LoggingOptions

- ListAuditFindings
- ListAuditMitigationActionsTasks
- ListMitigationActions
- ListRolePolicies
- ListTopics
- PutRolePolicy
- SetLoggingOptions
- SetV2LoggingOptions
- StartAuditMitigationActionsTask
- StartOnDemandAuditTask
- UpdateAccountAuditConfiguration

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with machine learning feature stores

The following code example shows how to:

- Set up IAM permissions
- Create a SageMaker execution role
- Create feature groups
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon SageMaker Feature Store Tutorial Script - Version 3
# This script demonstrates how to use Amazon SageMaker Feature Store with AWS CLI

# Setup logging
LOG_FILE="sagemaker-featurestore-tutorial.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting SageMaker Feature Store tutorial script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"
echo ""

# Track created resources for cleanup
CREATED_RESOURCES=()

# Function to handle errors
handle_error() {
    echo "ERROR: $1"
    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Function to check command status
check_status() {
    if echo "$1" | grep -i "error" > /dev/null; then
        handle_error "$1"
    fi
}

# Function to wait for feature group to be created
wait_for_feature_group() {
    local feature_group_name=$1
    local status="Creating"

    echo "Waiting for feature group ${feature_group_name} to be created..."

    while [ "$status" = "Creating" ]; do
        sleep 5
        status=$(aws sagemaker describe-feature-group \
            --feature-group-name "${feature_group_name}" \
            --query 'FeatureGroupStatus' \
```

```
            --output text)
        echo "Current status: ${status}"

        if [ "$status" = "Failed" ]; then
            handle_error "Feature group ${feature_group_name} creation failed"
        fi
    done

    echo "Feature group ${feature_group_name} is now ${status}"
}

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources..."

    # Clean up in reverse order
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        resource="${CREATED_RESOURCES[$i]}"
        resource_type=$(echo "$resource" | cut -d: -f1)
        resource_name=$(echo "$resource" | cut -d: -f2)

        echo "Deleting $resource_type: $resource_name"

        case "$resource_type" in
            "FeatureGroup")
                aws sagemaker delete-feature-group --feature-group-name
 "$resource_name"
                ;;
            "S3Bucket")
                echo "Emptying S3 bucket: $resource_name"
                aws s3 rm "s3://$resource_name" --recursive 2>/dev/null
                echo "Deleting S3 bucket: $resource_name"
                aws s3api delete-bucket --bucket "$resource_name" 2>/dev/null
                ;;
            "IAMRole")
                echo "Detaching policies from role: $resource_name"
                aws iam detach-role-policy --role-name "$resource_name" --policy-
arn "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess" 2>/dev/null
                aws iam detach-role-policy --role-name "$resource_name" --policy-
arn "arn:aws:iam::aws:policy/AmazonS3FullAccess" 2>/dev/null
                echo "Deleting IAM role: $resource_name"
                aws iam delete-role --role-name "$resource_name" 2>/dev/null
                ;;
            *)
```

```
                echo "Unknown resource type: $resource_type"
                ;;
        esac
    done
}

# Function to create SageMaker execution role
create_sagemaker_role() {
    local role_name="SageMakerFeatureStoreRole-$(openssl rand -hex 4)"

    echo "Creating SageMaker execution role: $role_name" >&2

    # Create trust policy document
    local trust_policy='{
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "sagemaker.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }'

    # Create the role
    local role_result=$(aws iam create-role \
        --role-name "$role_name" \
        --assume-role-policy-document "$trust_policy" \
        --description "SageMaker execution role for Feature Store tutorial" 2>&1)

    if echo "$role_result" | grep -i "error" > /dev/null; then
        handle_error "Failed to create IAM role: $role_result"
    fi

    echo "Role created successfully" >&2
    CREATED_RESOURCES+=("IAMRole:$role_name")

    # Tag the role
    echo "Tagging IAM role..." >&2
    aws iam tag-role --role-name "$role_name" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=sagemaker-featurestore 2>&1
```

```
    # Attach necessary policies
    echo "Attaching policies to role..." >&2

    # SageMaker execution policy
    local policy1_result=$(aws iam attach-role-policy \
        --role-name "$role_name" \
        --policy-arn "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess" 2>&1)

    if echo "$policy1_result" | grep -i "error" > /dev/null; then
        handle_error "Failed to attach SageMaker policy: $policy1_result"
    fi

    # S3 access policy
    local policy2_result=$(aws iam attach-role-policy \
        --role-name "$role_name" \
        --policy-arn "arn:aws:iam::aws:policy/AmazonS3FullAccess" 2>&1)

    if echo "$policy2_result" | grep -i "error" > /dev/null; then
        handle_error "Failed to attach S3 policy: $policy2_result"
    fi

    # Get account ID for role ARN
    local account_id=$(aws sts get-caller-identity --query Account --output text)
    local role_arn="arn:aws:iam::${account_id}:role/${role_name}"

    echo "Role ARN: $role_arn" >&2
    echo "Waiting 10 seconds for role to propagate..." >&2
    sleep 10

    # Return only the role ARN to stdout
    echo "$role_arn"
}

# Handle SageMaker execution role
ROLE_ARN=""

if [ -z "$1" ]; then
    echo "Creating SageMaker execution role automatically..."
    ROLE_ARN=$(create_sagemaker_role)
    if [ -z "$ROLE_ARN" ]; then
        handle_error "Failed to create SageMaker execution role"
    fi
else
    ROLE_ARN="$1"
```

```
    # Validate the role ARN
    ROLE_NAME=$(echo "$ROLE_ARN" | sed 's/.*role\///')
    ROLE_CHECK=$(aws iam get-role --role-name "$ROLE_NAME" 2>&1)
    if echo "$ROLE_CHECK" | grep -i "error" > /dev/null; then
        echo "Creating a new role automatically..."
        ROLE_ARN=$(create_sagemaker_role)
        if [ -z "$ROLE_ARN" ]; then
            handle_error "Failed to create SageMaker execution role"
        fi
    fi
fi

# Handle cleanup option
AUTO_CLEANUP=""
if [ -n "$2" ]; then
    AUTO_CLEANUP="$2"
fi

# Generate a random identifier for resource names
RANDOM_ID=$(openssl rand -hex 4)
echo "Using random identifier: $RANDOM_ID"

# Set variables
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
check_status "$ACCOUNT_ID"
echo "Account ID: $ACCOUNT_ID"

# Get current region
REGION=$(aws configure get region)
if [ -z "$REGION" ]; then
    REGION="us-east-1"
    echo "No default region configured, using: $REGION"
else
    echo "Using region: $REGION"
fi
# Check for shared prereq bucket
PREREQ_BUCKET=$(aws cloudformation describe-stacks --stack-name tutorial-prereqs-
bucket \
    --query 'Stacks[0].Outputs[?OutputKey==`BucketName`].OutputValue' --output
 text 2>/dev/null)
if [ -n "$PREREQ_BUCKET" ] && [ "$PREREQ_BUCKET" != "None" ]; then
    S3_BUCKET_NAME="$PREREQ_BUCKET"
    BUCKET_IS_SHARED=true
```

```
    echo "Using shared bucket: $S3_BUCKET_NAME"
else
    BUCKET_IS_SHARED=false
    S3_BUCKET_NAME="sagemaker-featurestore-${RANDOM_ID}-${ACCOUNT_ID}"
fi
PREFIX="featurestore-tutorial"
CURRENT_TIME=$(date +%s)

echo "Creating S3 bucket: $S3_BUCKET_NAME"
# Create bucket in current region (skip if using shared bucket)
if [ "$BUCKET_IS_SHARED" = "false" ]; then
if [ "$REGION" = "us-east-1" ]; then
    BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME" \
        --region "$REGION" 2>&1)
else
    BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME" \
        --region "$REGION" \
        --create-bucket-configuration LocationConstraint="$REGION" 2>&1)
fi

if echo "$BUCKET_RESULT" | grep -i "error" > /dev/null; then
    echo "Failed to create S3 bucket: $BUCKET_RESULT"
    exit 1
fi

echo "$BUCKET_RESULT"
CREATED_RESOURCES+=("S3Bucket:$S3_BUCKET_NAME")

# Tag the S3 bucket
echo "Tagging S3 bucket: $S3_BUCKET_NAME"
aws s3api put-bucket-tagging --bucket "$S3_BUCKET_NAME" --tagging
 'TagSet=[{Key=project,Value=doc-smith},{Key=tutorial,Value=sagemaker-
featurestore}]' 2>&1

# Block public access to the bucket
BLOCK_RESULT=$(aws s3api put-public-access-block \
    --bucket "$S3_BUCKET_NAME" \
    --public-access-block-configuration
 "BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets
 2>&1)

if echo "$BLOCK_RESULT" | grep -i "error" > /dev/null; then
    echo "Failed to block public access to S3 bucket: $BLOCK_RESULT"
    cleanup_resources
```

```
    exit 1
fi
else
    echo "Using shared bucket (skipping creation)"
fi

# Create feature groups
echo "Creating feature groups..."

# Create customers feature group
CUSTOMERS_FEATURE_GROUP_NAME="customers-feature-group-${RANDOM_ID}"
echo "Creating customers feature group: $CUSTOMERS_FEATURE_GROUP_NAME"

CUSTOMERS_RESPONSE=$(aws sagemaker create-feature-group \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record-identifier-feature-name "customer_id" \
    --event-time-feature-name "EventTime" \
    --feature-definitions '[
        {"FeatureName": "customer_id", "FeatureType": "Integral"},
        {"FeatureName": "name", "FeatureType": "String"},
        {"FeatureName": "age", "FeatureType": "Integral"},
        {"FeatureName": "address", "FeatureType": "String"},
        {"FeatureName": "membership_type", "FeatureType": "String"},
        {"FeatureName": "EventTime", "FeatureType": "Fractional"}
    ]' \
    --online-store-config '{"EnableOnlineStore": true}' \
    --offline-store-config '{
        "S3StorageConfig": {
            "S3Uri": "s3://'${S3_BUCKET_NAME}'/'${PREFIX}'"
        },
        "DisableGlueTableCreation": false
    }' \
    --role-arn "$ROLE_ARN" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=sagemaker-featurestore
 2>&1)

if echo "$CUSTOMERS_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to create customers feature group: $CUSTOMERS_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMERS_RESPONSE"
CREATED_RESOURCES+=("FeatureGroup:$CUSTOMERS_FEATURE_GROUP_NAME")
```

```
# Create orders feature group
ORDERS_FEATURE_GROUP_NAME="orders-feature-group-${RANDOM_ID}"
echo "Creating orders feature group: $ORDERS_FEATURE_GROUP_NAME"

ORDERS_RESPONSE=$(aws sagemaker create-feature-group \
    --feature-group-name "$ORDERS_FEATURE_GROUP_NAME" \
    --record-identifier-feature-name "customer_id" \
    --event-time-feature-name "EventTime" \
    --feature-definitions '[
        {"FeatureName": "customer_id", "FeatureType": "Integral"},
        {"FeatureName": "order_id", "FeatureType": "String"},
        {"FeatureName": "order_date", "FeatureType": "String"},
        {"FeatureName": "product", "FeatureType": "String"},
        {"FeatureName": "quantity", "FeatureType": "Integral"},
        {"FeatureName": "amount", "FeatureType": "Fractional"},
        {"FeatureName": "EventTime", "FeatureType": "Fractional"}
    ]' \
    --online-store-config '{"EnableOnlineStore": true}' \
    --offline-store-config '{
        "S3StorageConfig": {
            "S3Uri": "s3://'${S3_BUCKET_NAME}'/'${PREFIX}'"
        },
        "DisableGlueTableCreation": false
    }' \
    --role-arn "$ROLE_ARN" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=sagemaker-featurestore
 2>&1)

if echo "$ORDERS_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to create orders feature group: $ORDERS_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$ORDERS_RESPONSE"
CREATED_RESOURCES+=("FeatureGroup:$ORDERS_FEATURE_GROUP_NAME")

# Wait for feature groups to be created
wait_for_feature_group "$CUSTOMERS_FEATURE_GROUP_NAME"
wait_for_feature_group "$ORDERS_FEATURE_GROUP_NAME"

# Ingest data into feature groups
echo "Ingesting data into feature groups..."
```

```
# Ingest customer data
echo "Ingesting customer data..."
CUSTOMER1_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record '[
        {"FeatureName": "customer_id", "ValueAsString": "573291"},
        {"FeatureName": "name", "ValueAsString": "John Doe"},
        {"FeatureName": "age", "ValueAsString": "35"},
        {"FeatureName": "address", "ValueAsString": "123 Main St"},
        {"FeatureName": "membership_type", "ValueAsString": "premium"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$CUSTOMER1_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest customer 1 data: $CUSTOMER1_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMER1_RESPONSE"

CUSTOMER2_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record '[
        {"FeatureName": "customer_id", "ValueAsString": "109382"},
        {"FeatureName": "name", "ValueAsString": "Jane Smith"},
        {"FeatureName": "age", "ValueAsString": "28"},
        {"FeatureName": "address", "ValueAsString": "456 Oak Ave"},
        {"FeatureName": "membership_type", "ValueAsString": "standard"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$CUSTOMER2_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest customer 2 data: $CUSTOMER2_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMER2_RESPONSE"

# Ingest order data
echo "Ingesting order data..."
ORDER1_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
```

```
    --feature-group-name "$ORDERS_FEATURE_GROUP_NAME" \
    --record '[
        {"FeatureName": "customer_id", "ValueAsString": "573291"},
        {"FeatureName": "order_id", "ValueAsString": "ORD-001"},
        {"FeatureName": "order_date", "ValueAsString": "2023-01-15"},
        {"FeatureName": "product", "ValueAsString": "Laptop"},
        {"FeatureName": "quantity", "ValueAsString": "1"},
        {"FeatureName": "amount", "ValueAsString": "1299.99"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$ORDER1_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest order 1 data: $ORDER1_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$ORDER1_RESPONSE"

ORDER2_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
    --feature-group-name "$ORDERS_FEATURE_GROUP_NAME" \
    --record '[
        {"FeatureName": "customer_id", "ValueAsString": "109382"},
        {"FeatureName": "order_id", "ValueAsString": "ORD-002"},
        {"FeatureName": "order_date", "ValueAsString": "2023-01-20"},
        {"FeatureName": "product", "ValueAsString": "Smartphone"},
        {"FeatureName": "quantity", "ValueAsString": "1"},
        {"FeatureName": "amount", "ValueAsString": "899.99"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$ORDER2_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest order 2 data: $ORDER2_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$ORDER2_RESPONSE"

# Retrieve records from feature groups
echo "Retrieving records from feature groups..."

# Get a single customer record
echo "Getting customer record with ID 573291:"
```

```
CUSTOMER_RECORD=$(aws sagemaker-featurestore-runtime get-record \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record-identifier-value-as-string "573291" 2>&1)

if echo "$CUSTOMER_RECORD" | grep -i "error" > /dev/null; then
    echo "Failed to get customer record: $CUSTOMER_RECORD"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMER_RECORD"

# Get multiple records using batch-get-record
echo "Getting multiple records using batch-get-record:"
BATCH_RECORDS=$(aws sagemaker-featurestore-runtime batch-get-record \
    --identifiers '[
        {
            "FeatureGroupName": "'${CUSTOMERS_FEATURE_GROUP_NAME}'",
            "RecordIdentifiersValueAsString": ["573291", "109382"]
        },
        {
            "FeatureGroupName": "'${ORDERS_FEATURE_GROUP_NAME}'",
            "RecordIdentifiersValueAsString": ["573291", "109382"]
        }
    ]' 2>&1)

if echo "$BATCH_RECORDS" | grep -i "error" > /dev/null && ! echo "$BATCH_RECORDS"
 | grep -i "Records" > /dev/null; then
    echo "Failed to get batch records: $BATCH_RECORDS"
    cleanup_resources
    exit 1
fi

echo "$BATCH_RECORDS"

# List feature groups
echo "Listing feature groups:"
FEATURE_GROUPS=$(aws sagemaker list-feature-groups 2>&1)

if echo "$FEATURE_GROUPS" | grep -i "error" > /dev/null; then
    echo "Failed to list feature groups: $FEATURE_GROUPS"
    cleanup_resources
    exit 1
fi
```

```
echo "$FEATURE_GROUPS"

# Display summary of created resources
echo ""
echo "============================================"
echo "TUTORIAL COMPLETED SUCCESSFULLY!"
echo "============================================"
echo "Resources created:"
echo "- S3 Bucket: $S3_BUCKET_NAME"
echo "- Customers Feature Group: $CUSTOMERS_FEATURE_GROUP_NAME"
echo "- Orders Feature Group: $ORDERS_FEATURE_GROUP_NAME"
if [[ " ${CREATED_RESOURCES[@]} " =~ " IAMRole:" ]]; then
    echo "- IAM Role: $(echo "${CREATED_RESOURCES[@]}" | grep -o 'IAMRole:
[^[:space:]]*' | cut -d: -f2)"
fi
echo ""
echo "You can now:"
echo "1. View your feature groups in the SageMaker console"
echo "2. Query the offline store using Amazon Athena"
echo "3. Use the feature groups in your ML workflows"
echo "============================================"
echo ""

# Handle cleanup
if [ "$AUTO_CLEANUP" = "y" ]; then
    echo "Auto-cleanup enabled. Starting cleanup..."
    cleanup_resources
    echo "Cleanup completed."
elif [ "$AUTO_CLEANUP" = "n" ]; then
    echo "Auto-cleanup disabled. Resources will remain in your account."
    echo "To clean up later, run this script again with cleanup option 'y'"
else
    echo "============================================"
    echo "CLEANUP CONFIRMATION"
    echo "============================================"
    echo "Do you want to clean up all created resources? (y/n): "
    read -r CLEANUP_CHOICE

    if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
        echo "Starting cleanup..."
        cleanup_resources
        echo "Cleanup completed."
    else
```

```
        echo "Skipping cleanup. Resources will remain in your account."
        echo "To clean up later, delete the following resources:"
        echo "- Feature Groups: $CUSTOMERS_FEATURE_GROUP_NAME,
 $ORDERS_FEATURE_GROUP_NAME"
        echo "- S3 Bucket: $S3_BUCKET_NAME"
        if [[ " ${CREATED_RESOURCES[@]} " =~ " IAMRole:" ]]; then
            echo "- IAM Role: $(echo "${CREATED_RESOURCES[@]}" | grep -o
 'IAMRole:[^[:space:]]*' | cut -d: -f2)"
        fi
        echo ""
        echo "Estimated ongoing cost: ~$0.01 per month for online store"
    fi
fi

echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateBucket
  - CreateFeatureGroup
  - CreateRole
  - DeleteBucket
  - DeleteFeatureGroup
  - DeleteRole
  - DescribeFeatureGroup
  - DetachRolePolicy
  - GetCallerIdentity
  - GetRole
  - ListFeatureGroups
  - PutPublicAccessBlock
  - Rm

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with managed kubernetes clusters

The following code example shows how to:

- Create a VPC for your EKS cluster
- Create IAM roles for your EKS cluster
- Create your EKS cluster
- Configure kubectl to communicate with your cluster
- Create a managed node group
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon EKS Cluster Creation Script (v2)
# This script creates an Amazon EKS cluster with a managed node group using the
 AWS CLI

# Set up logging
LOG_FILE="eks-cluster-creation-v2.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon EKS cluster creation script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Error handling function
handle_error() {
    echo "ERROR: $1"
    echo "Attempting to clean up resources..."
    cleanup_resources
```

```
    exit 1
}

# Function to check command success
check_command() {
    if [ $? -ne 0 ] || echo "$1" | grep -i "error" > /dev/null; then
        handle_error "$1"
    fi
}

# Function to check if kubectl is installed
check_kubectl() {
    if ! command -v kubectl &> /dev/null; then
        echo "WARNING: kubectl is not installed or not in your PATH."
        echo ""
        echo "To install kubectl, follow these instructions based on your
 operating system:"
        echo ""
        echo "For Linux:"
        echo "  1. Download the latest release:"
        echo "     curl -LO \"https://dl.k8s.io/release/\$(curl -L -s https://
dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl\""
        echo ""
        echo "  2. Make the kubectl binary executable:"
        echo "     chmod +x ./kubectl"
        echo ""
        echo "  3. Move the binary to your PATH:"
        echo "     sudo mv ./kubectl /usr/local/bin/kubectl"
        echo ""
        echo "For macOS:"
        echo "  1. Using Homebrew:"
        echo "     brew install kubectl"
        echo "     or"
        echo "  2. Using curl:"
        echo "     curl -LO \"https://dl.k8s.io/release/\$(curl -L -s https://
dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl\""
        echo "     chmod +x ./kubectl"
        echo "     sudo mv ./kubectl /usr/local/bin/kubectl"
        echo ""
        echo "For Windows:"
        echo "  1. Using curl:"
        echo "     curl -LO \"https://dl.k8s.io/release/v1.28.0/bin/windows/
amd64/kubectl.exe\""
        echo "     Add the binary to your PATH"
```

```
        echo "     or"
        echo "  2. Using Chocolatey:"
        echo "     choco install kubernetes-cli"
        echo ""
        echo "After installation, verify with: kubectl version --client"
        echo ""
        return 1
    fi
    return 0
}

# Generate a random identifier for resource names
RANDOM_ID=$(LC_ALL=C tr -dc 'a-z0-9' < /dev/urandom | fold -w 6 | head -n 1)
STACK_NAME="eks-vpc-stack-${RANDOM_ID}"
CLUSTER_NAME="eks-cluster-${RANDOM_ID}"
NODEGROUP_NAME="eks-nodegroup-${RANDOM_ID}"
CLUSTER_ROLE_NAME="EKSClusterRole-${RANDOM_ID}"
NODE_ROLE_NAME="EKSNodeRole-${RANDOM_ID}"

echo "Using the following resource names:"
echo "- VPC Stack: $STACK_NAME"
echo "- EKS Cluster: $CLUSTER_NAME"
echo "- Node Group: $NODEGROUP_NAME"
echo "- Cluster IAM Role: $CLUSTER_ROLE_NAME"
echo "- Node IAM Role: $NODE_ROLE_NAME"

# Array to track created resources for cleanup
declare -a CREATED_RESOURCES

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources in reverse order..."

    # Check if node group exists and delete it
    if aws eks list-nodegroups --cluster-name "$CLUSTER_NAME" --query
 "nodegroups[?contains(@,'$NODEGROUP_NAME')]" --output text 2>/dev/null | grep -q
 "$NODEGROUP_NAME"; then
        echo "Deleting node group: $NODEGROUP_NAME"
        aws eks delete-nodegroup --cluster-name "$CLUSTER_NAME" --nodegroup-name
 "$NODEGROUP_NAME"
        echo "Waiting for node group deletion to complete..."
        aws eks wait nodegroup-deleted --cluster-name "$CLUSTER_NAME" --
nodegroup-name "$NODEGROUP_NAME"
        echo "Node group deleted successfully."
```

```
    fi

    # Check if cluster exists and delete it
    if aws eks describe-cluster --name "$CLUSTER_NAME" 2>/dev/null; then
        echo "Deleting cluster: $CLUSTER_NAME"
        aws eks delete-cluster --name "$CLUSTER_NAME"
        echo "Waiting for cluster deletion to complete (this may take several
 minutes)..."
        aws eks wait cluster-deleted --name "$CLUSTER_NAME"
        echo "Cluster deleted successfully."
    fi

    # Check if CloudFormation stack exists and delete it
    if aws cloudformation describe-stacks --stack-name "$STACK_NAME" 2>/dev/null;
 then
        echo "Deleting CloudFormation stack: $STACK_NAME"
        aws cloudformation delete-stack --stack-name "$STACK_NAME"
        echo "Waiting for CloudFormation stack deletion to complete..."
        aws cloudformation wait stack-delete-complete --stack-name "$STACK_NAME"
        echo "CloudFormation stack deleted successfully."
    fi

    # Clean up IAM roles
    if aws iam get-role --role-name "$NODE_ROLE_NAME" 2>/dev/null; then
        echo "Detaching policies from node role: $NODE_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSWorkerNodePolicy --role-name "$NODE_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly --role-name "$NODE_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKS_CNI_Policy --role-name "$NODE_ROLE_NAME"
        echo "Deleting node role: $NODE_ROLE_NAME"
        aws iam delete-role --role-name "$NODE_ROLE_NAME"
        echo "Node role deleted successfully."
    fi

    if aws iam get-role --role-name "$CLUSTER_ROLE_NAME" 2>/dev/null; then
        echo "Detaching policies from cluster role: $CLUSTER_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name "$CLUSTER_ROLE_NAME"
        echo "Deleting cluster role: $CLUSTER_ROLE_NAME"
        aws iam delete-role --role-name "$CLUSTER_ROLE_NAME"
        echo "Cluster role deleted successfully."
    fi
```

```
    echo "Cleanup complete."
}

# Trap to ensure cleanup on script exit
trap 'echo "Script interrupted. Cleaning up resources..."; cleanup_resources;
 exit 1' SIGINT SIGTERM

# Verify AWS CLI configuration
echo "Verifying AWS CLI configuration..."
AWS_ACCOUNT_INFO=$(aws sts get-caller-identity)
check_command "$AWS_ACCOUNT_INFO"
echo "AWS CLI is properly configured."

# Step 1: Create VPC using CloudFormation
echo "Step 1: Creating VPC with CloudFormation..."
echo "Creating CloudFormation stack: $STACK_NAME"

# Create the CloudFormation stack
CF_CREATE_OUTPUT=$(aws cloudformation create-stack \
  --stack-name "$STACK_NAME" \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=eks-gs)
check_command "$CF_CREATE_OUTPUT"
CREATED_RESOURCES+=("CloudFormation Stack: $STACK_NAME")

echo "Waiting for CloudFormation stack to complete (this may take a few
 minutes)..."
aws cloudformation wait stack-create-complete --stack-name "$STACK_NAME"
if [ $? -ne 0 ]; then
    handle_error "CloudFormation stack creation failed"
fi
echo "CloudFormation stack created successfully."

# Step 2: Create IAM roles for EKS
echo "Step 2: Creating IAM roles for EKS..."

# Create cluster role trust policy
echo "Creating cluster role trust policy..."
cat > eks-cluster-role-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create cluster role
echo "Creating cluster IAM role: $CLUSTER_ROLE_NAME"
CLUSTER_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$CLUSTER_ROLE_NAME" \
  --assume-role-policy-document file://"eks-cluster-role-trust-policy.json")
check_command "$CLUSTER_ROLE_OUTPUT"
aws iam tag-role --role-name "$CLUSTER_ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=eks-gs
CREATED_RESOURCES+=("IAM Role: $CLUSTER_ROLE_NAME")

# Attach policy to cluster role
echo "Attaching EKS cluster policy to role..."
ATTACH_CLUSTER_POLICY_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name "$CLUSTER_ROLE_NAME")
check_command "$ATTACH_CLUSTER_POLICY_OUTPUT"

# Create node role trust policy
echo "Creating node role trust policy..."
cat > node-role-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

```
# Create node role
echo "Creating node IAM role: $NODE_ROLE_NAME"
NODE_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$NODE_ROLE_NAME" \
  --assume-role-policy-document file://"node-role-trust-policy.json")
check_command "$NODE_ROLE_OUTPUT"
aws iam tag-role --role-name "$NODE_ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=eks-gs
CREATED_RESOURCES+=("IAM Role: $NODE_ROLE_NAME")

# Attach policies to node role
echo "Attaching EKS node policies to role..."
ATTACH_NODE_POLICY1_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name "$NODE_ROLE_NAME")
check_command "$ATTACH_NODE_POLICY1_OUTPUT"

ATTACH_NODE_POLICY2_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name "$NODE_ROLE_NAME")
check_command "$ATTACH_NODE_POLICY2_OUTPUT"

ATTACH_NODE_POLICY3_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name "$NODE_ROLE_NAME")
check_command "$ATTACH_NODE_POLICY3_OUTPUT"

# Step 3: Get VPC and subnet information
echo "Step 3: Getting VPC and subnet information..."

VPC_ID=$(aws cloudformation describe-stacks \
  --stack-name "$STACK_NAME" \
  --query "Stacks[0].Outputs[?OutputKey=='VpcId'].OutputValue" \
  --output text)
if [ -z "$VPC_ID" ]; then
    handle_error "Failed to get VPC ID from CloudFormation stack"
fi
echo "VPC ID: $VPC_ID"

SUBNET_IDS=$(aws cloudformation describe-stacks \
  --stack-name "$STACK_NAME" \
  --query "Stacks[0].Outputs[?OutputKey=='SubnetIds'].OutputValue" \
  --output text)
```

```
if [ -z "$SUBNET_IDS" ]; then
    handle_error "Failed to get Subnet IDs from CloudFormation stack"
fi
echo "Subnet IDs: $SUBNET_IDS"

SECURITY_GROUP_ID=$(aws cloudformation describe-stacks \
  --stack-name "$STACK_NAME" \
  --query "Stacks[0].Outputs[?OutputKey=='SecurityGroups'].OutputValue" \
  --output text)
if [ -z "$SECURITY_GROUP_ID" ]; then
    handle_error "Failed to get Security Group ID from CloudFormation stack"
fi
echo "Security Group ID: $SECURITY_GROUP_ID"

# Step 4: Create EKS cluster
echo "Step 4: Creating EKS cluster: $CLUSTER_NAME"

CLUSTER_ROLE_ARN=$(aws iam get-role --role-name "$CLUSTER_ROLE_NAME" --query
 "Role.Arn" --output text)
if [ -z "$CLUSTER_ROLE_ARN" ]; then
    handle_error "Failed to get Cluster Role ARN"
fi

echo "Creating EKS cluster (this will take 10-15 minutes)..."
CREATE_CLUSTER_OUTPUT=$(aws eks create-cluster \
  --name "$CLUSTER_NAME" \
  --role-arn "$CLUSTER_ROLE_ARN" \
  --resources-vpc-config
 subnetIds="$SUBNET_IDS",securityGroupIds="$SECURITY_GROUP_ID" \
  --tags Key=project,Value=doc-smith,Key=tutorial,Value=eks-gs)
check_command "$CREATE_CLUSTER_OUTPUT"
CREATED_RESOURCES+=("EKS Cluster: $CLUSTER_NAME")

echo "Waiting for EKS cluster to become active (this may take 10-15 minutes)..."
aws eks wait cluster-active --name "$CLUSTER_NAME"
if [ $? -ne 0 ]; then
    handle_error "Cluster creation failed or timed out"
fi
echo "EKS cluster is now active."

# Step 5: Configure kubectl
echo "Step 5: Configuring kubectl to communicate with the cluster..."

# Check if kubectl is installed
```

```
if ! check_kubectl; then
    echo "Will skip kubectl configuration steps but continue with the script."
    echo "You can manually configure kubectl later with: aws eks update-
kubeconfig --name \"$CLUSTER_NAME\""
else
    UPDATE_KUBECONFIG_OUTPUT=$(aws eks update-kubeconfig --name "$CLUSTER_NAME")
    check_command "$UPDATE_KUBECONFIG_OUTPUT"
    echo "kubectl configured successfully."

    # Test kubectl configuration
    echo "Testing kubectl configuration..."
    KUBECTL_TEST_OUTPUT=$(kubectl get svc 2>&1)
    if [ $? -ne 0 ]; then
        echo "Warning: kubectl configuration test failed. This might be due to
 permissions or network issues."
        echo "Error details: $KUBECTL_TEST_OUTPUT"
        echo "Continuing with script execution..."
    else
        echo "$KUBECTL_TEST_OUTPUT"
        echo "kubectl configuration test successful."
    fi
fi

# Step 6: Create managed node group
echo "Step 6: Creating managed node group: $NODEGROUP_NAME"

NODE_ROLE_ARN=$(aws iam get-role --role-name "$NODE_ROLE_NAME" --query "Role.Arn"
 --output text)
if [ -z "$NODE_ROLE_ARN" ]; then
    handle_error "Failed to get Node Role ARN"
fi

# Convert comma-separated subnet IDs to space-separated for the create-nodegroup
 command
SUBNET_IDS_ARRAY=(${SUBNET_IDS//,/ })

echo "Creating managed node group (this will take 5-10 minutes)..."
CREATE_NODEGROUP_OUTPUT=$(aws eks create-nodegroup \
  --cluster-name "$CLUSTER_NAME" \
  --nodegroup-name "$NODEGROUP_NAME" \
  --node-role "$NODE_ROLE_ARN" \
  --subnets "${SUBNET_IDS_ARRAY[@]}" \
  --tags Key=project,Value=doc-smith,Key=tutorial,Value=eks-gs)
check_command "$CREATE_NODEGROUP_OUTPUT"
```

```
CREATED_RESOURCES+=("EKS Node Group: $NODEGROUP_NAME")

echo "Waiting for node group to become active (this may take 5-10 minutes)..."
aws eks wait nodegroup-active --cluster-name "$CLUSTER_NAME" --nodegroup-name
 "$NODEGROUP_NAME"
if [ $? -ne 0 ]; then
    handle_error "Node group creation failed or timed out"
fi
echo "Node group is now active."

# Step 7: Verify nodes
echo "Step 7: Verifying nodes..."
echo "Waiting for nodes to register with the cluster (this may take a few
 minutes)..."
sleep 60  # Give nodes more time to register

# Check if kubectl is installed before attempting to use it
if ! check_kubectl; then
    echo "Cannot verify nodes without kubectl. Skipping this step."
    echo "You can manually verify nodes after installing kubectl with: kubectl
 get nodes"
else
    NODES_OUTPUT=$(kubectl get nodes 2>&1)
    if [ $? -ne 0 ]; then
        echo "Warning: Unable to get nodes. This might be due to permissions or
 the nodes are still registering."
        echo "Error details: $NODES_OUTPUT"
        echo "Continuing with script execution..."
    else
        echo "$NODES_OUTPUT"
        echo "Nodes verified successfully."
    fi
fi

# Step 8: View resources
echo "Step 8: Viewing cluster resources..."

echo "Cluster information:"
CLUSTER_INFO=$(aws eks describe-cluster --name "$CLUSTER_NAME")
echo "$CLUSTER_INFO"

echo "Node group information:"
NODEGROUP_INFO=$(aws eks describe-nodegroup --cluster-name "$CLUSTER_NAME" --
nodegroup-name "$NODEGROUP_NAME")
```

```
echo "$NODEGROUP_INFO"

echo "Kubernetes resources:"
if ! check_kubectl; then
    echo "Cannot list Kubernetes resources without kubectl. Skipping this step."
    echo "You can manually list resources after installing kubectl with: kubectl
 get all --all-namespaces"
else
    KUBE_RESOURCES=$(kubectl get all --all-namespaces 2>&1)
    if [ $? -ne 0 ]; then
        echo "Warning: Unable to get Kubernetes resources. This might be due to
 permissions."
        echo "Error details: $KUBE_RESOURCES"
        echo "Continuing with script execution..."
    else
        echo "$KUBE_RESOURCES"
    fi
fi

# Display summary of created resources
echo ""
echo "========================================="
echo "RESOURCES CREATED"
echo "========================================="
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "- $resource"
done
echo "========================================="

# Prompt for cleanup
echo ""
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    cleanup_resources
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later."
    echo "To clean up resources, run the following commands:"
```

```
    echo "1. Delete node group: aws eks delete-nodegroup --cluster-name
 $CLUSTER_NAME --nodegroup-name $NODEGROUP_NAME"
    echo "2. Wait for node group deletion: aws eks wait nodegroup-deleted --
cluster-name $CLUSTER_NAME --nodegroup-name $NODEGROUP_NAME"
    echo "3. Delete cluster: aws eks delete-cluster --name $CLUSTER_NAME"
    echo "4. Wait for cluster deletion: aws eks wait cluster-deleted --name
 $CLUSTER_NAME"
    echo "5. Delete CloudFormation stack: aws cloudformation delete-stack --
stack-name $STACK_NAME"
    echo "6. Detach and delete IAM roles for the node group and cluster"
fi

echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateCluster
  - CreateNodegroup
  - CreateRole
  - CreateStack
  - DeleteCluster
  - DeleteNodegroup
  - DeleteRole
  - DeleteStack
  - DescribeCluster
  - DescribeNodegroup
  - DescribeStacks
  - DetachRolePolicy
  - GetCallerIdentity
  - GetRole
  - ListNodegroups
  - UpdateKubeconfig
  - Wait (CloudFormation)
  - Wait (Amazon EKS)

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with managed streaming

The following code example shows how to:

- Create an MSK cluster
- Create IAM permissions for MSK access
- Create a client machine
- Get bootstrap brokers
- Set up the client machine
- Create a topic and produce/consume data
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon MSK Getting Started Tutorial Script - Version 8
# This script automates the steps in the Amazon MSK Getting Started tutorial
# It creates an MSK cluster, sets up IAM permissions, creates a client machine,
# and configures the client to interact with the cluster

# Set up logging
LOG_FILE="msk_tutorial_$(date +%Y%m%d_%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon MSK Getting Started Tutorial Script - Version 8"
echo "Logging to $LOG_FILE"
```

```
echo "============================================="

# Function to handle errors
handle_error() {
    echo "ERROR: $1"
    echo "Resources created so far:"
    if [ -n "$CLUSTER_ARN" ]; then echo "- MSK Cluster: $CLUSTER_ARN"; fi
    if [ -n "$POLICY_ARN" ]; then echo "- IAM Policy: $POLICY_ARN"; fi
    if [ -n "$ROLE_NAME" ]; then echo "- IAM Role: $ROLE_NAME"; fi
    if [ -n "$INSTANCE_PROFILE_NAME" ]; then echo "- IAM Instance Profile:
 $INSTANCE_PROFILE_NAME"; fi
    if [ -n "$CLIENT_SG_ID" ]; then echo "- Client Security Group:
 $CLIENT_SG_ID"; fi
    if [ -n "$INSTANCE_ID" ]; then echo "- EC2 Instance: $INSTANCE_ID"; fi
    if [ -n "$KEY_NAME" ]; then echo "- Key Pair: $KEY_NAME"; fi

    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Function to check if a resource exists
resource_exists() {
    local resource_type="$1"
    local resource_id="$2"

    case "$resource_type" in
        "cluster")
            aws kafka describe-cluster --cluster-arn "$resource_id" &>/dev/null
            ;;
        "policy")
            aws iam get-policy --policy-arn "$resource_id" &>/dev/null
            ;;
        "role")
            aws iam get-role --role-name "$resource_id" &>/dev/null
            ;;
        "instance-profile")
            aws iam get-instance-profile --instance-profile-name "$resource_id"
 &>/dev/null
            ;;
        "security-group")
            aws ec2 describe-security-groups --group-ids "$resource_id" &>/dev/
null
            ;;
```

```
        "instance")
            aws ec2 describe-instances --instance-ids "$resource_id" --query
 'Reservations[0].Instances[0].State.Name' --output text | grep -v "terminated"
 &>/dev/null
            ;;
        "key-pair")
            aws ec2 describe-key-pairs --key-names "$resource_id" &>/dev/null
            ;;
    esac
}

# Function to remove security group references
remove_security_group_references() {
    local sg_id="$1"

    if [ -z "$sg_id" ]; then
        echo "No security group ID provided for reference removal"
        return
    fi

    echo "Removing security group references for $sg_id"

    # Get all security groups in the VPC that might reference our client security
 group
    local vpc_security_groups=$(aws ec2 describe-security-groups \
        --filters "Name=vpc-id,Values=$DEFAULT_VPC_ID" \
        --query 'SecurityGroups[].GroupId' \
        --output text 2>/dev/null)

    if [ -n "$vpc_security_groups" ]; then
        for other_sg in $vpc_security_groups; do
            if [ "$other_sg" != "$sg_id" ]; then
                echo "Checking security group $other_sg for references to $sg_id"

                # Get the security group details in JSON format
                local sg_details=$(aws ec2 describe-security-groups \
                    --group-ids "$other_sg" \
                    --output json 2>/dev/null)

                if [ -n "$sg_details" ]; then
                    # Check if our security group is referenced in inbound rules
                    local has_inbound_ref=$(echo "$sg_details" | grep -o
 "\"GroupId\": \"$sg_id\"" | head -1)
```

```
                    if [ -n "$has_inbound_ref" ]; then
                        echo "Found inbound rules in $other_sg referencing
 $sg_id, removing them..."

                        # Try to remove common rule types
                        echo "Attempting to remove all-traffic rule"
                        aws ec2 revoke-security-group-ingress \
                            --group-id "$other_sg" \
                            --protocol all \
                            --source-group "$sg_id" 2>/dev/null || echo "No all-
traffic rule to remove"

                        # Try to remove TCP rules on common ports
                        for port in 22 80 443 9092 9094 9096; do
                            aws ec2 revoke-security-group-ingress \
                                --group-id "$other_sg" \
                                --protocol tcp \
                                --port "$port" \
                                --source-group "$sg_id" 2>/dev/null || true
                        done

                        # Try to remove UDP rules
                        aws ec2 revoke-security-group-ingress \
                            --group-id "$other_sg" \
                            --protocol udp \
                            --source-group "$sg_id" 2>/dev/null || true
                    fi

                    # Check for outbound rules (less common but possible)
                    local has_outbound_ref=$(echo "$sg_details" | grep -A 20
 "IpPermissionsEgress" | grep -o "\"GroupId\": \"$sg_id\"" | head -1)

                    if [ -n "$has_outbound_ref" ]; then
                        echo "Found outbound rules in $other_sg referencing
 $sg_id, removing them..."

                        aws ec2 revoke-security-group-egress \
                            --group-id "$other_sg" \
                            --protocol all \
                            --source-group "$sg_id" 2>/dev/null || echo "No
 outbound all-traffic rule to remove"
                    fi
                fi
            fi
```

```
        done
    fi

    echo "Completed security group reference removal for $sg_id"
}

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources..."

    # Delete EC2 instance if it exists
    if [ -n "$INSTANCE_ID" ] && resource_exists "instance" "$INSTANCE_ID"; then
        echo "Terminating EC2 instance: $INSTANCE_ID"
        aws ec2 terminate-instances --instance-ids "$INSTANCE_ID" || echo "Failed
 to terminate instance"
        echo "Waiting for instance to terminate..."
        aws ec2 wait instance-terminated --instance-ids "$INSTANCE_ID" || echo
 "Failed to wait for instance termination"
    fi

    # Delete MSK cluster first (to remove dependencies on security group)
    if [ -n "$CLUSTER_ARN" ] && resource_exists "cluster" "$CLUSTER_ARN"; then
        echo "Deleting MSK cluster: $CLUSTER_ARN"
        aws kafka delete-cluster --cluster-arn "$CLUSTER_ARN" || echo "Failed to
 delete cluster"

        # Wait a bit for the cluster deletion to start
        echo "Waiting 30 seconds for cluster deletion to begin..."
        sleep 30
    fi

    # Remove security group references before attempting deletion
    if [ -n "$CLIENT_SG_ID" ] && resource_exists "security-group"
 "$CLIENT_SG_ID"; then
        remove_security_group_references "$CLIENT_SG_ID"

        echo "Deleting security group: $CLIENT_SG_ID"
        # Try multiple times with longer delays to ensure dependencies are
 removed
        for i in {1..10}; do
            if aws ec2 delete-security-group --group-id "$CLIENT_SG_ID"; then
                echo "Security group deleted successfully"
                break
            fi
```

```
            echo "Failed to delete security group (attempt $i/10), retrying in 30
seconds..."
            sleep 30
        done
    fi

    # Delete key pair if it exists
    if [ -n "$KEY_NAME" ] && resource_exists "key-pair" "$KEY_NAME"; then
        echo "Deleting key pair: $KEY_NAME"
        aws ec2 delete-key-pair --key-name "$KEY_NAME" || echo "Failed to delete
key pair"
    fi

    # Remove role from instance profile
    if [ -n "$ROLE_NAME" ] && [ -n "$INSTANCE_PROFILE_NAME" ] && resource_exists
"instance-profile" "$INSTANCE_PROFILE_NAME"; then
        echo "Removing role from instance profile"
        aws iam remove-role-from-instance-profile \
            --instance-profile-name "$INSTANCE_PROFILE_NAME" \
            --role-name "$ROLE_NAME" || echo "Failed to remove role from instance
profile"
    fi

    # Delete instance profile
    if [ -n "$INSTANCE_PROFILE_NAME" ] && resource_exists "instance-profile"
"$INSTANCE_PROFILE_NAME"; then
        echo "Deleting instance profile: $INSTANCE_PROFILE_NAME"
        aws iam delete-instance-profile \
            --instance-profile-name "$INSTANCE_PROFILE_NAME" || echo "Failed to
delete instance profile"
    fi

    # Detach policy from role
    if [ -n "$ROLE_NAME" ] && [ -n "$POLICY_ARN" ] && resource_exists "role"
"$ROLE_NAME"; then
        echo "Detaching policy from role"
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "$POLICY_ARN" || echo "Failed to detach policy"
    fi

    # Delete role
    if [ -n "$ROLE_NAME" ] && resource_exists "role" "$ROLE_NAME"; then
        echo "Deleting role: $ROLE_NAME"
```

```
        aws iam delete-role --role-name "$ROLE_NAME" || echo "Failed to delete
 role"
    fi

    # Delete policy
    if [ -n "$POLICY_ARN" ] && resource_exists "policy" "$POLICY_ARN"; then
        echo "Deleting policy: $POLICY_ARN"
        aws iam delete-policy --policy-arn "$POLICY_ARN" || echo "Failed to
 delete policy"
    fi

    echo "Cleanup completed"
}

# Function to find a suitable subnet and instance type combination
find_suitable_subnet_and_instance_type() {
    local vpc_id="$1"
    local -a subnet_array=("${!2}")

    # List of instance types to try, in order of preference
    local instance_types=("t3.micro" "t2.micro" "t3.small" "t2.small")

    echo "Finding suitable subnet and instance type combination..."

    for instance_type in "${instance_types[@]}"; do
        echo "Trying instance type: $instance_type"

        for subnet_id in "${subnet_array[@]}"; do
            # Get the availability zone for this subnet
            local az=$(aws ec2 describe-subnets \
                --subnet-ids "$subnet_id" \
                --query 'Subnets[0].AvailabilityZone' \
                --output text)

            echo "  Checking subnet $subnet_id in AZ $az"

            # Check if this instance type is available in this AZ
            local available=$(aws ec2 describe-instance-type-offerings \
                --location-type availability-zone \
                --filters "Name=location,Values=$az" "Name=instance-type,Values=
$instance_type" \
                --query 'InstanceTypeOfferings[0].InstanceType' \
                --output text 2>/dev/null)
```

```
            if [ "$available" = "$instance_type" ]; then
                echo "  # Found suitable combination: $instance_type in $az
 (subnet: $subnet_id)"
                SELECTED_SUBNET_ID="$subnet_id"
                SELECTED_INSTANCE_TYPE="$instance_type"
                return 0
            else
                echo "  # $instance_type not available in $az"
            fi
        done
    done

    echo "ERROR: Could not find any suitable subnet and instance type
 combination"
    return 1
}

# Generate unique identifiers
RANDOM_SUFFIX=$(LC_ALL=C tr -dc 'a-z0-9' < /dev/urandom | fold -w 8 | head -n 1)
CLUSTER_NAME="MSKTutorialCluster-${RANDOM_SUFFIX}"
POLICY_NAME="msk-tutorial-policy-${RANDOM_SUFFIX}"
ROLE_NAME="msk-tutorial-role-${RANDOM_SUFFIX}"
INSTANCE_PROFILE_NAME="msk-tutorial-profile-${RANDOM_SUFFIX}"
SG_NAME="MSKClientSecurityGroup-${RANDOM_SUFFIX}"

echo "Using the following resource names:"
echo "- Cluster Name: $CLUSTER_NAME"
echo "- Policy Name: $POLICY_NAME"
echo "- Role Name: $ROLE_NAME"
echo "- Instance Profile Name: $INSTANCE_PROFILE_NAME"
echo "- Security Group Name: $SG_NAME"
echo "================================================"

# Step 1: Create an MSK Provisioned cluster
echo "Step 1: Creating MSK Provisioned cluster"

# Get the default VPC ID first
echo "Getting default VPC..."
DEFAULT_VPC_ID=$(aws ec2 describe-vpcs \
    --filters "Name=is-default,Values=true" \
    --query "Vpcs[0].VpcId" \
    --output text)

if [ -z "$DEFAULT_VPC_ID" ] || [ "$DEFAULT_VPC_ID" = "None" ]; then
```

```
    handle_error "Could not find default VPC. Please ensure you have a default
 VPC in your region."
fi

echo "Default VPC ID: $DEFAULT_VPC_ID"

# Get available subnets in the default VPC
echo "Getting available subnets in the default VPC..."
SUBNETS=$(aws ec2 describe-subnets \
    --filters "Name=vpc-id,Values=$DEFAULT_VPC_ID" "Name=default-for-
az,Values=true" \
    --query "Subnets[0:3].SubnetId" \
    --output text)

# Convert space-separated subnet IDs to an array
read -r -a SUBNET_ARRAY <<< "$SUBNETS"

if [ ${#SUBNET_ARRAY[@]} -lt 3 ]; then
    handle_error "Not enough subnets available in the default VPC. Need at least
 3 subnets, found ${#SUBNET_ARRAY[@]}."
fi

# Get default security group for the default VPC
echo "Getting default security group for the default VPC..."
DEFAULT_SG=$(aws ec2 describe-security-groups \
    --filters "Name=group-name,Values=default" "Name=vpc-id,Values=
$DEFAULT_VPC_ID" \
    --query "SecurityGroups[0].GroupId" \
    --output text)

if [ -z "$DEFAULT_SG" ] || [ "$DEFAULT_SG" = "None" ]; then
    handle_error "Could not find default security group for VPC $DEFAULT_VPC_ID"
fi

echo "Creating MSK cluster: $CLUSTER_NAME"
echo "Using VPC: $DEFAULT_VPC_ID"
echo "Using subnets: ${SUBNET_ARRAY[0]} ${SUBNET_ARRAY[1]} ${SUBNET_ARRAY[2]}"
echo "Using security group: $DEFAULT_SG"

# Create the MSK cluster with proper error handling
CLUSTER_RESPONSE=$(aws kafka create-cluster \
    --cluster-name "$CLUSTER_NAME" \
```

```
    --broker-node-group-info "{\"InstanceType\": \"kafka.t3.small\",
 \"ClientSubnets\": [\"${SUBNET_ARRAY[0]}\", \"${SUBNET_ARRAY[1]}\",
 \"${SUBNET_ARRAY[2]}\"], \"SecurityGroups\": [\"$DEFAULT_SG\"]}" \
    --kafka-version "3.6.0" \
    --number-of-broker-nodes 3 \
    --encryption-info "{\"EncryptionInTransit\": {\"InCluster\": true,
 \"ClientBroker\": \"TLS\"}}" \
    --tags project=doc-smith,tutorial=amazon-managed-streaming-for-apache-kafka-
gs 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create MSK cluster: $CLUSTER_RESPONSE"
fi

# Extract the cluster ARN using grep
CLUSTER_ARN=$(echo "$CLUSTER_RESPONSE" | grep -o '"ClusterArn": "[^"]*' | cut -
d'"' -f4)

if [ -z "$CLUSTER_ARN" ]; then
    handle_error "Failed to extract cluster ARN from response: $CLUSTER_RESPONSE"
fi

echo "MSK cluster creation initiated. ARN: $CLUSTER_ARN"
echo "Waiting for cluster to become active (this may take 15-20 minutes)..."

# Wait for the cluster to become active
while true; do
    CLUSTER_STATUS=$(aws kafka describe-cluster --cluster-arn "$CLUSTER_ARN" --
query "ClusterInfo.State" --output text 2>/dev/null)

    if [ $? -ne 0 ]; then
        echo "Failed to get cluster status. Retrying in 30 seconds..."
        sleep 30
        continue
    fi

    echo "Current cluster status: $CLUSTER_STATUS"

    if [ "$CLUSTER_STATUS" = "ACTIVE" ]; then
        echo "Cluster is now active!"
        break
    elif [ "$CLUSTER_STATUS" = "FAILED" ]; then
        handle_error "Cluster creation failed"
```

```
    fi

    echo "Still waiting for cluster to become active... (checking again in 60
 seconds)"
    sleep 60
done

echo "============================================="
# Step 2: Create an IAM role granting access to create topics on the Amazon MSK
 cluster
echo "Step 2: Creating IAM policy and role"

# Get account ID and region
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
REGION=$(aws configure get region)
if [ -z "$REGION" ]; then
    REGION=$(aws ec2 describe-availability-zones --query
 'AvailabilityZones[0].RegionName' --output text)
fi

if [ -z "$ACCOUNT_ID" ] || [ -z "$REGION" ]; then
    handle_error "Could not determine AWS account ID or region"
fi

echo "Account ID: $ACCOUNT_ID"
echo "Region: $REGION"

# Create IAM policy
echo "Creating IAM policy: $POLICY_NAME"
POLICY_DOCUMENT="{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Action\": [
                \"kafka-cluster:Connect\",
                \"kafka-cluster:AlterCluster\",
                \"kafka-cluster:DescribeCluster\"
            ],
            \"Resource\": [
                \"$CLUSTER_ARN\"
            ]
        },
        {
```

```
            \"Effect\": \"Allow\",
            \"Action\": [
                \"kafka-cluster:*Topic*\",
                \"kafka-cluster:WriteData\",
                \"kafka-cluster:ReadData\"
            ],
            \"Resource\": [
                \"arn:aws:kafka:$REGION:$ACCOUNT_ID:topic/$CLUSTER_NAME/*\"
            ]
        },
        {
            \"Effect\": \"Allow\",
            \"Action\": [
                \"kafka-cluster:AlterGroup\",
                \"kafka-cluster:DescribeGroup\"
            ],
            \"Resource\": [
                \"arn:aws:kafka:$REGION:$ACCOUNT_ID:group/$CLUSTER_NAME/*\"
            ]
        }
    ]
}"

POLICY_RESPONSE=$(aws iam create-policy \
    --policy-name "$POLICY_NAME" \
    --policy-document "$POLICY_DOCUMENT" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create IAM policy: $POLICY_RESPONSE"
fi

# Extract the policy ARN using grep
POLICY_ARN=$(echo "$POLICY_RESPONSE" | grep -o '"Arn": "[^"]*' | cut -d'"' -f4)

if [ -z "$POLICY_ARN" ]; then
    handle_error "Failed to extract policy ARN from response: $POLICY_RESPONSE"
fi

aws iam tag-policy --policy-arn "$POLICY_ARN" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=amazon-managed-
streaming-for-apache-kafka-gs

echo "IAM policy created. ARN: $POLICY_ARN"
```

```
# Create IAM role for EC2
echo "Creating IAM role: $ROLE_NAME"
TRUST_POLICY="{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow\",
\"Principal\":{\"Service\":\"ec2.amazonaws.com\"},\"Action\":\"sts:AssumeRole
\"}]}"

ROLE_RESPONSE=$(aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document "$TRUST_POLICY" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create IAM role: $ROLE_RESPONSE"
fi

echo "IAM role created: $ROLE_NAME"
aws iam tag-role --role-name "$ROLE_NAME" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=amazon-managed-
streaming-for-apache-kafka-gs

# Attach policy to role
echo "Attaching policy to role"
ATTACH_RESPONSE=$(aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "$POLICY_ARN" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to attach policy to role: $ATTACH_RESPONSE"
fi

echo "Policy attached to role"

# Create instance profile and add role to it
echo "Creating instance profile: $INSTANCE_PROFILE_NAME"
PROFILE_RESPONSE=$(aws iam create-instance-profile \
    --instance-profile-name "$INSTANCE_PROFILE_NAME" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create instance profile: $PROFILE_RESPONSE"
fi
```

```
echo "Instance profile created"

echo "Adding role to instance profile"
ADD_ROLE_RESPONSE=$(aws iam add-role-to-instance-profile \
    --instance-profile-name "$INSTANCE_PROFILE_NAME" \
    --role-name "$ROLE_NAME" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to add role to instance profile: $ADD_ROLE_RESPONSE"
fi

echo "Role added to instance profile"

# Wait a moment for IAM propagation
echo "Waiting 10 seconds for IAM propagation..."
sleep 10

echo "============================================="

# Step 3: Create a client machine
echo "Step 3: Creating client machine"

# Find a suitable subnet and instance type combination
if ! find_suitable_subnet_and_instance_type "$DEFAULT_VPC_ID" SUBNET_ARRAY[@];
 then
    handle_error "Could not find a suitable subnet and instance type combination"
fi

echo "Selected subnet: $SELECTED_SUBNET_ID"
echo "Selected instance type: $SELECTED_INSTANCE_TYPE"

# Verify the subnet is in the same VPC we're using
SUBNET_VPC_ID=$(aws ec2 describe-subnets \
    --subnet-ids "$SELECTED_SUBNET_ID" \
    --query 'Subnets[0].VpcId' \
    --output text)

if [ "$SUBNET_VPC_ID" != "$DEFAULT_VPC_ID" ]; then
    handle_error "Subnet VPC ($SUBNET_VPC_ID) does not match default VPC
 ($DEFAULT_VPC_ID)"
fi

echo "VPC ID: $SUBNET_VPC_ID"
```

```
# Get security group ID from the MSK cluster
echo "Getting security group ID from the MSK cluster"
MSK_SG_ID=$(aws kafka describe-cluster \
    --cluster-arn "$CLUSTER_ARN" \
    --query 'ClusterInfo.BrokerNodeGroupInfo.SecurityGroups[0]' \
    --output text)

if [ -z "$MSK_SG_ID" ] || [ "$MSK_SG_ID" = "None" ]; then
    handle_error "Failed to get security group ID from cluster"
fi

echo "MSK security group ID: $MSK_SG_ID"

# Create security group for client
echo "Creating security group for client: $SG_NAME"
CLIENT_SG_RESPONSE=$(aws ec2 create-security-group \
    --group-name "$SG_NAME" \
    --description "Security group for MSK client" \
    --vpc-id "$DEFAULT_VPC_ID" \
    --tag-specifications 'ResourceType=security-
group,Tags=[{Key=project,Value=doc-smith},{Key=tutorial,Value=amazon-managed-
streaming-for-apache-kafka-gs}]' 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create security group: $CLIENT_SG_RESPONSE"
fi

# Extract the security group ID using grep
CLIENT_SG_ID=$(echo "$CLIENT_SG_RESPONSE" | grep -o '"GroupId": "[^"]*' | cut -
d'"' -f4)

if [ -z "$CLIENT_SG_ID" ]; then
    handle_error "Failed to extract security group ID from response:
 $CLIENT_SG_RESPONSE"
fi

echo "Client security group created. ID: $CLIENT_SG_ID"

# Allow SSH access to client from your IP only
echo "Getting your public IP address"
MY_IP=$(curl -s https://checkip.amazonaws.com 2>/dev/null)
```

```
if [ -z "$MY_IP" ]; then
    echo "Warning: Could not determine your IP address. Using 0.0.0.0/0 (not
 recommended for production)"
    MY_IP="0.0.0.0/0"
else
    MY_IP="$MY_IP/32"
    echo "Your IP address: $MY_IP"
fi

echo "Adding SSH ingress rule to client security group"
SSH_RULE_RESPONSE=$(aws ec2 authorize-security-group-ingress \
    --group-id "$CLIENT_SG_ID" \
    --protocol tcp \
    --port 22 \
    --cidr "$MY_IP" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    echo "Warning: Failed to add SSH ingress rule: $SSH_RULE_RESPONSE"
    echo "You may need to manually add SSH access to security group
 $CLIENT_SG_ID"
fi

echo "SSH ingress rule added"

# Update MSK security group to allow traffic from client security group
echo "Adding ingress rule to MSK security group to allow traffic from client"
MSK_RULE_RESPONSE=$(aws ec2 authorize-security-group-ingress \
    --group-id "$MSK_SG_ID" \
    --protocol all \
    --source-group "$CLIENT_SG_ID" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    echo "Warning: Failed to add ingress rule to MSK security group:
 $MSK_RULE_RESPONSE"
    echo "You may need to manually add ingress rule to security group $MSK_SG_ID"
fi

echo "Ingress rule added to MSK security group"

# Create key pair
KEY_NAME="MSKKeyPair-${RANDOM_SUFFIX}"
echo "Creating key pair: $KEY_NAME"
```

```
KEY_RESPONSE=$(aws ec2 create-key-pair --key-name "$KEY_NAME" --tag-
specifications 'ResourceType=key-pair,Tags=[{Key=project,Value=doc-smith},
{Key=tutorial,Value=amazon-managed-streaming-for-apache-kafka-gs}]' --query
 'KeyMaterial' --output text 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create key pair: $KEY_RESPONSE"
fi

# Save the private key to a file
KEY_FILE="${KEY_NAME}.pem"
echo "$KEY_RESPONSE" > "$KEY_FILE"
chmod 400 "$KEY_FILE"

echo "Key pair created and saved to $KEY_FILE"

# Get the latest Amazon Linux 2 AMI
echo "Getting latest Amazon Linux 2 AMI ID"
AMI_ID=$(aws ec2 describe-images \
    --owners amazon \
    --filters "Name=name,Values=amzn2-ami-hvm-*-x86_64-gp2"
 "Name=state,Values=available" \
    --query "sort_by(Images, &CreationDate)[-1].ImageId" \
    --output text 2>/dev/null)

if [ -z "$AMI_ID" ] || [ "$AMI_ID" = "None" ]; then
    handle_error "Failed to get Amazon Linux 2 AMI ID"
fi

echo "Using AMI ID: $AMI_ID"

# Launch EC2 instance with the selected subnet and instance type
echo "Launching EC2 instance"
echo "Instance type: $SELECTED_INSTANCE_TYPE"
echo "Subnet: $SELECTED_SUBNET_ID"

INSTANCE_RESPONSE=$(aws ec2 run-instances \
    --image-id "$AMI_ID" \
    --instance-type "$SELECTED_INSTANCE_TYPE" \
    --key-name "$KEY_NAME" \
    --security-group-ids "$CLIENT_SG_ID" \
    --subnet-id "$SELECTED_SUBNET_ID" \
    --iam-instance-profile "Name=$INSTANCE_PROFILE_NAME" \
```

```
    --tag-specifications
 "ResourceType=instance,Tags=[{Key=Name,Value=MSKTutorialClient-
${RANDOM_SUFFIX}},{Key=project,Value=doc-smith},{Key=tutorial,Value=amazon-
managed-streaming-for-apache-kafka-gs}]" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to launch EC2 instance: $INSTANCE_RESPONSE"
fi

# Extract the instance ID using grep
INSTANCE_ID=$(echo "$INSTANCE_RESPONSE" | grep -o '"InstanceId": "[^"]*' | head
 -1 | cut -d'"' -f4)

if [ -z "$INSTANCE_ID" ]; then
    handle_error "Failed to extract instance ID from response:
 $INSTANCE_RESPONSE"
fi

echo "EC2 instance launched successfully. ID: $INSTANCE_ID"
echo "Waiting for instance to be running..."

# Wait for the instance to be running
aws ec2 wait instance-running --instance-ids "$INSTANCE_ID"

if [ $? -ne 0 ]; then
    handle_error "Instance failed to reach running state"
fi

# Wait a bit more for the instance to initialize
echo "Instance is running. Waiting 30 seconds for initialization..."
sleep 30

# Get public DNS name of instance
CLIENT_DNS=$(aws ec2 describe-instances \
    --instance-ids "$INSTANCE_ID" \
    --query 'Reservations[0].Instances[0].PublicDnsName' \
    --output text)

if [ -z "$CLIENT_DNS" ] || [ "$CLIENT_DNS" = "None" ]; then
    echo "Warning: Could not get public DNS name for instance. Trying public
 IP..."
    CLIENT_DNS=$(aws ec2 describe-instances \
        --instance-ids "$INSTANCE_ID" \
```

```
        --query 'Reservations[0].Instances[0].PublicIpAddress' \
        --output text)

    if [ -z "$CLIENT_DNS" ] || [ "$CLIENT_DNS" = "None" ]; then
        handle_error "Failed to get public DNS name or IP address for instance"
    fi
fi

echo "Client instance DNS/IP: $CLIENT_DNS"
echo "============================================="
# Get bootstrap brokers with improved logic
echo "Getting bootstrap brokers"
MAX_RETRIES=10
RETRY_COUNT=0
BOOTSTRAP_BROKERS=""
AUTH_METHOD=""

while [ -z "$BOOTSTRAP_BROKERS" ] || [ "$BOOTSTRAP_BROKERS" = "None" ]; do
    # Get the full bootstrap brokers response
    BOOTSTRAP_RESPONSE=$(aws kafka get-bootstrap-brokers \
        --cluster-arn "$CLUSTER_ARN" 2>/dev/null)

    if [ $? -eq 0 ] && [ -n "$BOOTSTRAP_RESPONSE" ]; then
        # Try to get IAM authentication brokers first using grep
        BOOTSTRAP_BROKERS=$(echo "$BOOTSTRAP_RESPONSE" | grep -o
 '"BootstrapBrokerStringSaslIam": "[^"]*' | cut -d'"' -f4)
        if [ -n "$BOOTSTRAP_BROKERS" ]; then
            AUTH_METHOD="IAM"
        else
            # Fall back to TLS authentication
            BOOTSTRAP_BROKERS=$(echo "$BOOTSTRAP_RESPONSE" | grep -o
 '"BootstrapBrokerStringTls": "[^"]*' | cut -d'"' -f4)
            if [ -n "$BOOTSTRAP_BROKERS" ]; then
                AUTH_METHOD="TLS"
            fi
        fi
    fi

    RETRY_COUNT=$((RETRY_COUNT + 1))

    if [ "$RETRY_COUNT" -ge "$MAX_RETRIES" ]; then
        echo "Warning: Could not get bootstrap brokers after $MAX_RETRIES
 attempts."
        echo "You may need to manually retrieve them later using:"
```

```
        echo "aws kafka get-bootstrap-brokers --cluster-arn $CLUSTER_ARN"
        BOOTSTRAP_BROKERS="BOOTSTRAP_BROKERS_NOT_AVAILABLE"
        AUTH_METHOD="UNKNOWN"
        break
    fi

    if [ -z "$BOOTSTRAP_BROKERS" ] || [ "$BOOTSTRAP_BROKERS" = "None" ]; then
        echo "Bootstrap brokers not available yet. Retrying in 30 seconds...
 (Attempt $RETRY_COUNT/$MAX_RETRIES)"
        sleep 30
    fi
done

echo "Bootstrap brokers: $BOOTSTRAP_BROKERS"
echo "Authentication method: $AUTH_METHOD"
echo "============================================="

# Create setup script for the client machine
echo "Creating setup script for the client machine"
cat > setup_client.sh << 'EOF'
#!/bin/bash

# Set up logging
LOG_FILE="client_setup_$(date +%Y%m%d_%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting client setup"
echo "============================================="

# Install Java
echo "Installing Java"
sudo yum -y install java-11

# Set environment variables
echo "Setting up environment variables"
export KAFKA_VERSION="3.6.0"
echo "KAFKA_VERSION=$KAFKA_VERSION"

# Download and extract Apache Kafka
echo "Downloading Apache Kafka"
wget https://archive.apache.org/dist/kafka/$KAFKA_VERSION/kafka_2.13-
$KAFKA_VERSION.tgz
if [ $? -ne 0 ]; then
    echo "Failed to download Kafka. Trying alternative mirror..."
```

```
    wget https://www.apache.org/dyn/closer.cgi?path=/kafka/$KAFKA_VERSION/
kafka_2.13-$KAFKA_VERSION.tgz
fi

echo "Extracting Kafka"
tar -xzf kafka_2.13-$KAFKA_VERSION.tgz
export KAFKA_ROOT=$(pwd)/kafka_2.13-$KAFKA_VERSION
echo "KAFKA_ROOT=$KAFKA_ROOT"

# Download the MSK IAM authentication package (needed for both IAM and TLS)
echo "Downloading MSK IAM authentication package"
cd $KAFKA_ROOT/libs
wget https://github.com/aws/aws-msk-iam-auth/releases/latest/download/aws-msk-
iam-auth-1.1.6-all.jar
if [ $? -ne 0 ]; then
    echo "Failed to download specific version. Trying to get latest version..."
    LATEST_VERSION=$(curl -s https://api.github.com/repos/aws/aws-msk-iam-auth/
releases/latest | grep -o '"tag_name": "[^"]*' | cut -d'"' -f4)
    wget https://github.com/aws/aws-msk-iam-auth/releases/download/
$LATEST_VERSION/aws-msk-iam-auth-$LATEST_VERSION-all.jar
    if [ $? -ne 0 ]; then
        echo "Failed to download IAM auth package. Please check the URL and try
 again."
        exit 1
    fi
    export CLASSPATH=$KAFKA_ROOT/libs/aws-msk-iam-auth-$LATEST_VERSION-all.jar
else
    export CLASSPATH=$KAFKA_ROOT/libs/aws-msk-iam-auth-1.1.6-all.jar
fi
echo "CLASSPATH=$CLASSPATH"

# Create client properties file based on authentication method
echo "Creating client properties file"
cd $KAFKA_ROOT/config

# The AUTH_METHOD_PLACEHOLDER will be replaced by the script
AUTH_METHOD="AUTH_METHOD_PLACEHOLDER"

if [ "$AUTH_METHOD" = "IAM" ]; then
    echo "Configuring for IAM authentication"
    cat > client.properties << 'EOT'
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=software.amazon.msk.auth.iam.IAMLoginModule required;
```

```
sasl.client.callback.handler.class=software.amazon.msk.auth.iam.IAMClientCallbackHandler
EOT
elif [ "$AUTH_METHOD" = "TLS" ]; then
    echo "Configuring for TLS authentication"
    cat > client.properties << 'EOT'
security.protocol=SSL
EOT
else
    echo "Unknown authentication method. Creating basic TLS configuration."
    cat > client.properties << 'EOT'
security.protocol=SSL
EOT
fi

echo "Client setup completed"
echo "============================================="

# Create a script to set up environment variables
cat > ~/setup_env.sh << 'EOT'
#!/bin/bash
export KAFKA_VERSION="3.6.0"
export KAFKA_ROOT=~/kafka_2.13-$KAFKA_VERSION
export CLASSPATH=$KAFKA_ROOT/libs/aws-msk-iam-auth-1.1.6-all.jar
export BOOTSTRAP_SERVER="BOOTSTRAP_SERVER_PLACEHOLDER"
export AUTH_METHOD="AUTH_METHOD_PLACEHOLDER"

echo "Environment variables set:"
echo "KAFKA_VERSION=$KAFKA_VERSION"
echo "KAFKA_ROOT=$KAFKA_ROOT"
echo "CLASSPATH=$CLASSPATH"
echo "BOOTSTRAP_SERVER=$BOOTSTRAP_SERVER"
echo "AUTH_METHOD=$AUTH_METHOD"
EOT

chmod +x ~/setup_env.sh

echo "Created environment setup script: ~/setup_env.sh"
echo "Run 'source ~/setup_env.sh' to set up your environment"
EOF

# Replace placeholders in the setup script
if [ -n "$BOOTSTRAP_BROKERS" ] && [ "$BOOTSTRAP_BROKERS" != "None" ] &&
 [ "$BOOTSTRAP_BROKERS" != "BOOTSTRAP_BROKERS_NOT_AVAILABLE" ]; then
    sed -i "s|BOOTSTRAP_SERVER_PLACEHOLDER|$BOOTSTRAP_BROKERS|g" setup_client.sh
```

```
else
    # If bootstrap brokers are not available, provide instructions to get them
    sed -i "s|BOOTSTRAP_SERVER_PLACEHOLDER|\$(aws kafka get-bootstrap-brokers
 --cluster-arn $CLUSTER_ARN --query 'BootstrapBrokerStringTls' --output text)|g"
 setup_client.sh
fi

# Replace auth method placeholder
sed -i "s|AUTH_METHOD_PLACEHOLDER|$AUTH_METHOD|g" setup_client.sh

echo "Setup script created"
echo "============================================="

# Display summary of created resources
echo ""
echo "============================================="
echo "RESOURCE SUMMARY"
echo "============================================="
echo "MSK Cluster ARN: $CLUSTER_ARN"
echo "MSK Cluster Name: $CLUSTER_NAME"
echo "Authentication Method: $AUTH_METHOD"
echo "IAM Policy ARN: $POLICY_ARN"
echo "IAM Role Name: $ROLE_NAME"
echo "IAM Instance Profile: $INSTANCE_PROFILE_NAME"
echo "Client Security Group: $CLIENT_SG_ID"
echo "EC2 Instance ID: $INSTANCE_ID"
echo "EC2 Instance Type: $SELECTED_INSTANCE_TYPE"
echo "EC2 Instance DNS: $CLIENT_DNS"
echo "Key Pair: $KEY_NAME (saved to $KEY_FILE)"
echo "Bootstrap Brokers: $BOOTSTRAP_BROKERS"
echo "============================================="
echo ""

# Instructions for connecting to the instance and setting up the client
echo "NEXT STEPS:"
echo "1. Connect to your EC2 instance:"
echo "   ssh -i $KEY_FILE ec2-user@$CLIENT_DNS"
echo ""
echo "2. Upload the setup script to your instance:"
echo "   scp -i $KEY_FILE setup_client.sh ec2-user@$CLIENT_DNS:~/"
echo ""
echo "3. Run the setup script on your instance:"
echo "   ssh -i $KEY_FILE ec2-user@$CLIENT_DNS 'chmod +x ~/setup_client.sh && ~/
setup_client.sh'"
```

```
echo ""
echo "4. Source the environment setup script:"
echo "   source ~/setup_env.sh"
echo ""

# Provide different instructions based on authentication method
if [ "$AUTH_METHOD" = "IAM" ]; then
    echo "5. Create a Kafka topic (using IAM authentication):"
    echo "   \$KAFKA_ROOT/bin/kafka-topics.sh --create \\"
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --command-config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --replication-factor 3 \\"
    echo "     --partitions 1 \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "6. Start a producer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-producer.sh \\"
    echo "     --broker-list \$BOOTSTRAP_SERVER \\"
    echo "     --producer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "7. Start a consumer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-consumer.sh \\"
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --consumer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic \\"
    echo "     --from-beginning"
elif [ "$AUTH_METHOD" = "TLS" ]; then
    echo "5. Create a Kafka topic (using TLS authentication):"
    echo "   \$KAFKA_ROOT/bin/kafka-topics.sh --create \\"
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --command-config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --replication-factor 3 \\"
    echo "     --partitions 1 \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "6. Start a producer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-producer.sh \\"
    echo "     --broker-list \$BOOTSTRAP_SERVER \\"
    echo "     --producer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "7. Start a consumer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-consumer.sh \\"
```

```
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --consumer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic \\"
    echo "     --from-beginning"
else
    echo "5. Manually retrieve bootstrap brokers and configure authentication:"
    echo "   aws kafka get-bootstrap-brokers --cluster-arn $CLUSTER_ARN"
fi

echo ""
echo "8. Verify the topic was created:"
echo "   \$KAFKA_ROOT/bin/kafka-topics.sh --list \\"
echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
echo "     --command-config \$KAFKA_ROOT/config/client.properties"
echo "================================================"
echo ""

# Ask if user wants to clean up resources
echo ""
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ $CLEANUP_CHOICE =~ ^[Yy]$ ]]; then
    cleanup_resources
    echo "All resources have been cleaned up."
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later."
    echo "To clean up resources, run this script again and choose 'y' when
 prompted."
fi

echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddRoleToInstanceProfile
  - AttachRolePolicy
  - AuthorizeSecurityGroupIngress

- CreateCluster
- CreateInstanceProfile
- CreateKeyPair
- CreatePolicy
- CreateRole
- CreateSecurityGroup
- DeleteCluster
- DeleteInstanceProfile
- DeleteKeyPair
- DeletePolicy
- DeleteRole
- DeleteSecurityGroup
- DescribeAvailabilityZones
- DescribeCluster
- DescribeImages
- DescribeInstanceTypeOfferings
- DescribeInstances
- DescribeKeyPairs
- DescribeSecurityGroups
- DescribeSubnets
- DescribeVpcs
- DetachRolePolicy
- GetBootstrapBrokers
- GetCallerIdentity
- GetInstanceProfile
- GetPolicy
- GetRole
- RemoveRoleFromInstanceProfile
- RevokeSecurityGroupEgress
- RevokeSecurityGroupIngress

- RunInstances
- TerminateInstances
- Wait

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with provisioned data warehouse clusters

The following code example shows how to:

- Create a Redshift cluster
- Create an IAM role for S3 access
- Create tables and load data
- Run example queries
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon Redshift Provisioned Cluster Tutorial Script
# This script creates a Redshift cluster, loads sample data, runs queries, and
 cleans up resources
# Version 4: Security improvements and best practices

set -euo pipefail

# Set up logging
```

```
LOG_FILE="redshift_tutorial.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon Redshift tutorial script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Function to handle errors
handle_error() {
    echo "ERROR: $1" >&2
    echo "Resources created so far:"
    if [ -n "${CLUSTER_ID:-}" ]; then echo "- Redshift Cluster: $CLUSTER_ID"; fi
    if [ -n "${ROLE_NAME:-}" ]; then echo "- IAM Role: $ROLE_NAME"; fi

    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources..."

    # Delete the cluster if it exists
    if [ -n "${CLUSTER_ID:-}" ]; then
        echo "Deleting Redshift cluster: $CLUSTER_ID"
        aws redshift delete-cluster --cluster-identifier "$CLUSTER_ID" --skip-
final-cluster-snapshot 2>/dev/null || true
        echo "Waiting for cluster deletion to complete..."
        aws redshift wait cluster-deleted --cluster-identifier "$CLUSTER_ID" 2>/
dev/null || true
        echo "Cluster deleted successfully."
    fi

    # Delete the IAM role if it exists
    if [ -n "${ROLE_NAME:-}" ]; then
        echo "Removing IAM role policy..."
        aws iam delete-role-policy --role-name "$ROLE_NAME" --policy-name
 RedshiftS3Access 2>/dev/null || true

        echo "Deleting IAM role: $ROLE_NAME"
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi

    # Clean up temporary files
```

```
    rm -f redshift-trust-policy.json redshift-s3-policy.json

    echo "Cleanup completed."
}

# Trap errors and cleanup
trap 'handle_error "Script interrupted"' INT TERM

# Function to wait for SQL statement to complete
wait_for_statement() {
    local statement_id=$1
    local max_attempts=30
    local attempt=1
    local status=""

    echo "Waiting for statement $statement_id to complete..."

    while [ $attempt -le $max_attempts ]; do
        status=$(aws redshift-data describe-statement --id "$statement_id" --
query 'Status' --output text 2>/dev/null || echo "")

        if [ "$status" == "FINISHED" ]; then
            echo "Statement completed successfully."
            return 0
        elif [ "$status" == "FAILED" ]; then
            local error=$(aws redshift-data describe-statement --id
 "$statement_id" --query 'Error' --output text 2>/dev/null || echo "Unknown
 error")
            echo "Statement failed with error: $error" >&2
            return 1
        elif [ "$status" == "ABORTED" ]; then
            echo "Statement was aborted." >&2
            return 1
        fi

        echo "Statement status: $status. Waiting... (Attempt $attempt/
$max_attempts)"
        sleep 10
        ((attempt++))
    done

    echo "Timed out waiting for statement to complete." >&2
    return 1
}
```

```
# Function to check if IAM role is attached to cluster
check_role_attached() {
    local role_arn=$1
    local max_attempts=10
    local attempt=1

    echo "Checking if IAM role is attached to the cluster..."

    while [ $attempt -le $max_attempts ]; do
        local status=$(aws redshift describe-clusters \
            --cluster-identifier "$CLUSTER_ID" \
            --query "Clusters[0].IamRoles[?IamRoleArn=='$role_arn'].ApplyStatus"
 \
            --output text 2>/dev/null || echo "")

        if [ "$status" == "in-sync" ]; then
            echo "IAM role is successfully attached to the cluster."
            return 0
        fi

        echo "IAM role status: $status. Waiting... (Attempt $attempt/
$max_attempts)"
        sleep 30
        ((attempt++))
    done

    echo "Timed out waiting for IAM role to be attached." >&2
    return 1
}

# Validate required commands
for cmd in aws jq; do
    if ! command -v "$cmd" &> /dev/null; then
        handle_error "Required command '$cmd' not found. Please install it and
 try again."
    fi
done

# Validate AWS credentials
if ! aws sts get-caller-identity &>/dev/null; then
    handle_error "AWS credentials not configured or invalid"
fi
```

```
# Variables to track created resources
CLUSTER_ID="examplecluster"
ROLE_NAME="RedshiftS3Role-$(date +%s)"
DB_NAME="dev"
DB_USER="awsuser"

# Generate secure password using AWS Secrets Manager or random string
if command -v openssl &> /dev/null; then
    DB_PASSWORD=$(openssl rand -base64 32 | tr -d "=+/" | cut -c1-20)
else
    DB_PASSWORD="TempPass$(date +%s | md5sum | cut -c1-20)"
fi

# Validate password meets requirements
if [ ${#DB_PASSWORD} -lt 8 ]; then
    handle_error "Generated password does not meet minimum length requirement"
fi

# Store password securely (optional: use AWS Secrets Manager in production)
echo "Generated database password (store securely): $DB_PASSWORD"

echo "=== Step 1: Creating Amazon Redshift Cluster ==="

# Create the Redshift cluster with encryption and audit logging enabled
echo "Creating Redshift cluster: $CLUSTER_ID"
CLUSTER_RESULT=$(aws redshift create-cluster \
  --cluster-identifier "$CLUSTER_ID" \
  --node-type ra3.xlplus \
  --number-of-nodes 2 \
  --master-username "$DB_USER" \
  --master-user-password "$DB_PASSWORD" \
  --db-name "$DB_NAME" \
  --port 5439 \
  --encrypted \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=redshift-provisioned \
  2>&1) || handle_error "Failed to create Redshift cluster"

echo "$CLUSTER_RESULT"
echo "Waiting for cluster to become available..."

# Wait for the cluster to be available
aws redshift wait cluster-available --cluster-identifier "$CLUSTER_ID" ||
 handle_error "Timeout waiting for cluster to become available"
```

```
# Get cluster status to confirm
CLUSTER_STATUS=$(aws redshift describe-clusters \
  --cluster-identifier "$CLUSTER_ID" \
  --query 'Clusters[0].ClusterStatus' \
  --output text)

echo "Cluster status: $CLUSTER_STATUS"

echo "=== Step 2: Creating IAM Role for S3 Access ==="

# Create trust policy file with restricted permissions
echo "Creating trust policy for Redshift"
cat > redshift-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "redshift.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

chmod 600 redshift-trust-policy.json

# Create IAM role
echo "Creating IAM role: $ROLE_NAME"
ROLE_RESULT=$(aws iam create-role \
  --role-name "$ROLE_NAME" \
  --assume-role-policy-document file://redshift-trust-policy.json 2>&1) ||
 handle_error "Failed to create IAM role"

echo "$ROLE_RESULT"

# Get the role ARN
ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn' --output
 text)
echo "Role ARN: $ROLE_ARN"

# Tag the IAM role
```

```
echo "Tagging IAM role: $ROLE_NAME"
aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=redshift-provisioned

# Create policy document for S3 access with principle of least privilege
echo "Creating S3 access policy"
cat > redshift-s3-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::redshift-downloads",
        "arn:aws:s3:::redshift-downloads/*"
      ]
    }
  ]
}
EOF

chmod 600 redshift-s3-policy.json

# Attach policy to role
echo "Attaching S3 access policy to role"
POLICY_RESULT=$(aws iam put-role-policy \
  --role-name "$ROLE_NAME" \
  --policy-name RedshiftS3Access \
  --policy-document file://redshift-s3-policy.json 2>&1) || handle_error "Failed
 to attach policy to role"

echo "$POLICY_RESULT"

# Attach role to cluster
echo "Attaching IAM role to Redshift cluster"
ATTACH_ROLE_RESULT=$(aws redshift modify-cluster-iam-roles \
  --cluster-identifier "$CLUSTER_ID" \
  --add-iam-roles "$ROLE_ARN" 2>&1) || handle_error "Failed to attach role to
 cluster"
```

```
echo "$ATTACH_ROLE_RESULT"

# Wait for the role to be attached
echo "Waiting for IAM role to be attached to the cluster..."
if ! check_role_attached "$ROLE_ARN"; then
    handle_error "Failed to attach IAM role to cluster"
fi

echo "=== Step 3: Getting Cluster Connection Information ==="

# Get cluster endpoint
CLUSTER_INFO=$(aws redshift describe-clusters \
  --cluster-identifier "$CLUSTER_ID" \
  --query 'Clusters[0].Endpoint.{Address:Address,Port:Port}' \
  --output json)

echo "Cluster endpoint information:"
echo "$CLUSTER_INFO"

echo "=== Step 4: Creating Tables and Loading Data ==="

echo "Creating sales table"
SALES_TABLE_ID=$(aws redshift-data execute-statement \
  --cluster-identifier "$CLUSTER_ID" \
  --database "$DB_NAME" \
  --db-user "$DB_USER" \
  --sql "DROP TABLE IF EXISTS sales; CREATE TABLE sales(salesid integer not null,
 listid integer not null distkey, sellerid integer not null, buyerid integer
 not null, eventid integer not null, dateid smallint not null sortkey, qtysold
 smallint not null, pricepaid decimal(8,2), commission decimal(8,2), saletime
 timestamp);" \
  --query 'Id' --output text)

echo "Sales table creation statement ID: $SALES_TABLE_ID"

# Wait for statement to complete
if ! wait_for_statement "$SALES_TABLE_ID"; then
    handle_error "Failed to create sales table"
fi

echo "Creating date table"
DATE_TABLE_ID=$(aws redshift-data execute-statement \
  --cluster-identifier "$CLUSTER_ID" \
  --database "$DB_NAME" \
```

```
  --db-user "$DB_USER" \
  --sql "DROP TABLE IF EXISTS date; CREATE TABLE date(dateid smallint not null
 distkey sortkey, caldate date not null, day character(3) not null, week smallint
 not null, month character(5) not null, qtr character(5) not null, year smallint
 not null, holiday boolean default('N'));" \
  --query 'Id' --output text)

echo "Date table creation statement ID: $DATE_TABLE_ID"

# Wait for statement to complete
if ! wait_for_statement "$DATE_TABLE_ID"; then
    handle_error "Failed to create date table"
fi

echo "Loading data into sales table"
SALES_LOAD_ID=$(aws redshift-data execute-statement \
  --cluster-identifier "$CLUSTER_ID" \
  --database "$DB_NAME" \
  --db-user "$DB_USER" \
  --sql "COPY sales FROM 's3://redshift-downloads/tickit/sales_tab.txt' DELIMITER
 '\t' TIMEFORMAT 'MM/DD/YYYY HH:MI:SS' REGION 'us-east-1' IAM_ROLE '$ROLE_ARN';"
 \
  --query 'Id' --output text)

echo "Sales data load statement ID: $SALES_LOAD_ID"

# Wait for statement to complete
if ! wait_for_statement "$SALES_LOAD_ID"; then
    handle_error "Failed to load data into sales table"
fi

echo "Loading data into date table"
DATE_LOAD_ID=$(aws redshift-data execute-statement \
  --cluster-identifier "$CLUSTER_ID" \
  --database "$DB_NAME" \
  --db-user "$DB_USER" \
  --sql "COPY date FROM 's3://redshift-downloads/tickit/date2008_pipe.txt'
 DELIMITER '|' REGION 'us-east-1' IAM_ROLE '$ROLE_ARN';" \
  --query 'Id' --output text)

echo "Date data load statement ID: $DATE_LOAD_ID"

# Wait for statement to complete
if ! wait_for_statement "$DATE_LOAD_ID"; then
```

```
    handle_error "Failed to load data into date table"
fi

echo "=== Step 5: Running Example Queries ==="

echo "Running query: Get definition for the sales table"
QUERY1_ID=$(aws redshift-data execute-statement \
  --cluster-identifier "$CLUSTER_ID" \
  --database "$DB_NAME" \
  --db-user "$DB_USER" \
  --sql "SELECT * FROM pg_table_def WHERE tablename = 'sales';" \
  --query 'Id' --output text)

echo "Query 1 statement ID: $QUERY1_ID"

# Wait for statement to complete
if ! wait_for_statement "$QUERY1_ID"; then
    handle_error "Query 1 failed"
fi

# Get and display results
echo "Query 1 results (first 10 rows):"
aws redshift-data get-statement-result --id "$QUERY1_ID" --max-items 10

echo "Running query: Find total sales on a given calendar date"
QUERY2_ID=$(aws redshift-data execute-statement \
  --cluster-identifier "$CLUSTER_ID" \
  --database "$DB_NAME" \
  --db-user "$DB_USER" \
  --sql "SELECT sum(qtysold) FROM sales, date WHERE sales.dateid = date.dateid
 AND caldate = '2008-01-05';" \
  --query 'Id' --output text)

echo "Query 2 statement ID: $QUERY2_ID"

# Wait for statement to complete
if ! wait_for_statement "$QUERY2_ID"; then
    handle_error "Query 2 failed"
fi

# Get and display results
echo "Query 2 results:"
aws redshift-data get-statement-result --id "$QUERY2_ID"
```

```
echo "=== Tutorial Complete ==="
echo "The following resources were created:"
echo "- Redshift Cluster: $CLUSTER_ID"
echo "- IAM Role: $ROLE_NAME"

echo ""
echo "================================================"
echo "CLEANUP CONFIRMATION"
echo "================================================"
echo "Cleaning up all created resources..."
cleanup_resources
echo "All resources have been cleaned up."

# Securely clear password from memory
DB_PASSWORD=""

echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - CreateCluster
  - CreateRole
  - DeleteCluster
  - DeleteRole
  - DeleteRolePolicy
  - DescribeClusters
  - GetRole
  - ModifyClusterIamRoles
  - PutRolePolicy
  - Wait

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with workflow orchestration

The following code example shows how to:

- Create an IAM role for Step Functions
- Create your first state machine
- Start your state machine execution
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS Step Functions Getting Started Tutorial Script
# This script creates and runs a Step Functions state machine based on the AWS
 Step Functions Getting Started tutorial

set -euo pipefail

# Security: Restrict umask to prevent unintended file permissions
umask 077

# Parse command line arguments
AUTO_CLEANUP=true
while [[ $# -gt 0 ]]; do
    case $1 in
        --auto-cleanup)
            AUTO_CLEANUP=true
            shift
            ;;
        -h|--help)
            echo "Usage: $0 [--auto-cleanup] [--help]"
            echo "  --auto-cleanup: Automatically clean up resources without
 prompting"
            echo "  --help: Show this help message"
            exit 0
```

```
            ;;
        *)
            echo "Unknown option: $1"
            echo "Use --help for usage information"
            exit 1
            ;;
    esac
done

# Set up logging with secure permissions
LOG_FILE="step-functions-tutorial.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"

# Security: Use process substitution with explicit FD cleanup
exec 3>&1 4>&2
exec > >(tee -a "$LOG_FILE") 2>&1
trap 'exec 1>&3 2>&4 3>&- 4>&-' EXIT

echo "Starting AWS Step Functions Getting Started Tutorial..."
echo "Logging to $LOG_FILE"

# Verify AWS CLI is installed and configured
if ! command -v aws &> /dev/null; then
    echo "ERROR: AWS CLI is not installed"
    exit 1
fi

# Verify AWS credentials are configured
if ! aws sts get-caller-identity &> /dev/null; then
    echo "ERROR: AWS credentials are not configured or invalid"
    exit 1
fi

# Check if jq is available for better JSON parsing
if ! command -v jq &> /dev/null; then
    echo "WARNING: jq is not installed. Using basic JSON parsing which may be
 less reliable."
    echo "Consider installing jq for better error handling: brew install jq
 (macOS) or apt-get install jq (Ubuntu)"
    USE_JQ=false
else
    USE_JQ=true
fi
```

```
# Use fixed region that supports Amazon Comprehend
CURRENT_REGION="us-west-2"
echo "Using fixed AWS region: $CURRENT_REGION (supports Amazon Comprehend)"

# Set AWS CLI to use the fixed region for all commands
export AWS_DEFAULT_REGION="$CURRENT_REGION"
export AWS_REGION="$CURRENT_REGION"

# Amazon Comprehend is available in us-west-2, so we can always enable it
echo "Amazon Comprehend is available in region $CURRENT_REGION"
SKIP_COMPREHEND=false

# Security: Initialize all resource variables
STATE_MACHINE_ARN=""
ROLE_NAME=""
ROLE_ARN=""
POLICY_ARN=""
STEPFUNCTIONS_POLICY_ARN=""
EXECUTION_ARN=""
EXECUTION2_ARN=""
EXECUTION3_ARN=""

# Performance: Cache for AWS API calls to reduce redundant requests
declare -A API_CACHE

# Function to make cached AWS CLI calls
aws_call_cached() {
    local cache_key="$1"
    shift

    if [[ -v API_CACHE["$cache_key"] ]]; then
        echo "${API_CACHE[$cache_key]}"
        return 0
    fi

    local result
    result=$(aws "$@" 2>&1) || return $?
    API_CACHE["$cache_key"]="$result"
    echo "$result"
}

# Function to check for API errors in JSON response with optimized jq usage
check_api_error() {
```

```
    local response="$1"
    local operation="$2"

    if [[ "$USE_JQ" == "true" ]]; then
        # Use jq for more reliable JSON parsing with efficient error detection
        if echo "$response" | jq -e '.Error // .error // empty' > /dev/null 2>&1;
 then
            local error_message=$(echo "$response" | jq -r
 '.Error.Message // .Error.Code // .error // "Unknown error"' 2>/dev/null)
            handle_error "$operation failed: $error_message"
        fi
    else
        # Fallback to grep-based detection with optimized pattern
        if echo "$response" | grep -qE '"[Ee]rror":|"error":'; then
            handle_error "$operation failed: $response"
        fi
    fi
}

# Function to extract JSON field efficiently
extract_json_field() {
    local json="$1"
    local field="$2"

    if [[ "$USE_JQ" == "true" ]]; then
        echo "$json" | jq -r "$field" 2>/dev/null
    else
        echo "$json" | grep -oP "\"${field}\":\s*\"\K[^\"]+|\"${field}\":\s*
\K[^,}]+" | head -1
    fi
}

# Function to securely wait for resource propagation with exponential backoff
wait_for_propagation() {
    local resource_type="$1"
    local wait_time="${2:-10}"

    # Validate wait_time is a positive integer
    if ! [[ "$wait_time" =~ ^[0-9]+$ ]] || [ "$wait_time" -lt 1 ] ||
 [ "$wait_time" -gt 300 ]; then
        echo "WARNING: Invalid wait time $wait_time, using default 10 seconds"
        wait_time=10
    fi
```

```
    echo "Waiting for $resource_type to propagate ($wait_time seconds)..."
    sleep "$wait_time"
}

# Function to validate JSON file efficiently
validate_json_file() {
    local file="$1"

    if [[ "$USE_JQ" == "true" ]]; then
        if ! jq empty "$file" 2>/dev/null; then
            handle_error "Invalid JSON in $file"
        fi
    fi
}

# Function to handle errors
handle_error() {
    echo "ERROR: $1"
    echo "Resources created:"
    if [ -n "${STATE_MACHINE_ARN:-}" ]; then
        echo "- State Machine: $STATE_MACHINE_ARN"
    fi
    if [ -n "${ROLE_NAME:-}" ]; then
        echo "- IAM Role: $ROLE_NAME"
    fi
    if [ -n "${POLICY_ARN:-}" ]; then
        echo "- IAM Policy: $POLICY_ARN"
    fi
    if [ -n "${STEPFUNCTIONS_POLICY_ARN:-}" ]; then
        echo "- Step Functions Policy: $STEPFUNCTIONS_POLICY_ARN"
    fi

    echo "Attempting to clean up resources..."
    cleanup
    exit 1
}

# Function to securely clean up resources with parallel deletion
cleanup() {
    echo "Cleaning up resources..."

    # Delete state machine if it exists
    if [ -n "${STATE_MACHINE_ARN:-}" ]; then
        echo "Deleting state machine: $STATE_MACHINE_ARN"
```

```
        aws stepfunctions delete-state-machine --state-machine-arn
 "$STATE_MACHINE_ARN" 2>/dev/null &
    fi

    # Detach and delete policies if they exist
    if [ -n "${POLICY_ARN:-}" ] && [ -n "${ROLE_NAME:-}" ]; then
        echo "Detaching Comprehend policy $POLICY_ARN from role $ROLE_NAME"
        aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$POLICY_ARN" 2>/dev/null &
    fi

    if [ -n "${STEPFUNCTIONS_POLICY_ARN:-}" ] && [ -n "${ROLE_NAME:-}" ]; then
        echo "Detaching Step Functions policy $STEPFUNCTIONS_POLICY_ARN from role
 $ROLE_NAME"
        aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$STEPFUNCTIONS_POLICY_ARN" 2>/dev/null &
    fi

    # Wait for detach operations to complete
    wait 2>/dev/null || true

    # Delete custom policies if they exist
    if [ -n "${POLICY_ARN:-}" ]; then
        echo "Deleting Comprehend policy: $POLICY_ARN"
        aws iam delete-policy --policy-arn "$POLICY_ARN" 2>/dev/null &
    fi

    if [ -n "${STEPFUNCTIONS_POLICY_ARN:-}" ]; then
        echo "Deleting Step Functions policy: $STEPFUNCTIONS_POLICY_ARN"
        aws iam delete-policy --policy-arn "$STEPFUNCTIONS_POLICY_ARN" 2>/dev/
null &
    fi

    # Wait for policy deletion to complete
    wait 2>/dev/null || true

    # Delete role if it exists
    if [ -n "${ROLE_NAME:-}" ]; then
        echo "Deleting role: $ROLE_NAME"
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || echo "Failed
 to delete role"
    fi

    # Remove temporary files securely
```

```
    echo "Removing temporary files"
    local temp_files=(
        "hello-world.json"
        "updated-hello-world.json"
        "sentiment-hello-world.json"
        "step-functions-trust-policy.json"
        "comprehend-policy.json"
        "stepfunctions-policy.json"
        "input.json"
        "sentiment-input.json"
    )

    for file in "${temp_files[@]}"; do
        if [ -f "$file" ]; then
            if command -v shred &> /dev/null; then
                shred -vfz -n 3 "$file" 2>/dev/null || rm -f "$file"
            else
                rm -f "$file"
            fi
        fi
    done
}

# Security: Set trap to cleanup on script exit
trap cleanup EXIT

# Generate a secure random identifier for resource names
RANDOM_ID=$(openssl rand -hex 4)
ROLE_NAME="StepFunctionsHelloWorldRole-${RANDOM_ID}"
POLICY_NAME="DetectSentimentPolicy-${RANDOM_ID}"
STATE_MACHINE_NAME="MyFirstStateMachine-${RANDOM_ID}"

echo "Using random identifier: $RANDOM_ID"
echo "Role name: $ROLE_NAME"
echo "Policy name: $POLICY_NAME"
echo "State machine name: $STATE_MACHINE_NAME"

# Step 1: Create the state machine definition
echo "Creating state machine definition..."
cat > hello-world.json << 'EOF'
{
  "Comment": "A Hello World example of the Amazon States Language using a Pass
 state",
  "StartAt": "SetVariables",
```

```
  "States": {
    "SetVariables": {
      "Type": "Pass",
      "Result": {
        "IsHelloWorldExample": true,
        "ExecutionWaitTimeInSeconds": 10
      },
      "Next": "IsHelloWorldExample"
    },
    "IsHelloWorldExample": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": true,
          "Next": "WaitState"
        }
      ],
      "Default": "FailState"
    },
    "WaitState": {
      "Type": "Wait",
      "SecondsPath": "$.ExecutionWaitTimeInSeconds",
      "Next": "ParallelProcessing"
    },
    "ParallelProcessing": {
      "Type": "Parallel",
      "Branches": [
        {
          "StartAt": "Process1",
          "States": {
            "Process1": {
              "Type": "Pass",
              "Result": {
                "message": "Processing task 1"
              },
              "End": true
            }
          }
        },
        {
          "StartAt": "Process2",
          "States": {
            "Process2": {
```

```
              "Type": "Pass",
              "Result": {
                "message": "Processing task 2"
              },
              "End": true
            }
          }
        }
      ],
      "Next": "CheckpointState"
    },
    "CheckpointState": {
      "Type": "Pass",
      "Result": {
        "CheckpointMessage": "Workflow completed successfully!"
      },
      "Next": "SuccessState"
    },
    "SuccessState": {
      "Type": "Succeed"
    },
    "FailState": {
      "Type": "Fail",
      "Error": "NotHelloWorldExample",
      "Cause": "The IsHelloWorldExample value was false"
    }
  }
}
EOF

# Create IAM role trust policy
echo "Creating IAM role trust policy..."
cat > step-functions-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "states.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
EOF

# Create IAM role
echo "Creating IAM role: $ROLE_NAME"
ROLE_RESULT=$(aws iam create-role \
  --role-name "$ROLE_NAME" \
  --assume-role-policy-document file://step-functions-trust-policy.json 2>&1)

check_api_error "$ROLE_RESULT" "Create IAM role"
echo "Role created successfully"
aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-step-functions-gs

# Get the role ARN
ROLE_ARN=$(extract_json_field "$ROLE_RESULT" ".Role.Arn")

if [ -z "$ROLE_ARN" ]; then
    handle_error "Failed to extract role ARN"
fi
echo "Role ARN: $ROLE_ARN"

# Create a custom policy for Step Functions with least privilege
echo "Creating custom policy for Step Functions..."
cat > stepfunctions-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution",
        "states:DescribeExecution",
        "states:StopExecution"
      ],
      "Resource": "arn:aws:states:*:*:stateMachine:*"
    }
  ]
}
EOF

# Create the policy
echo "Creating Step Functions policy..."
STEPFUNCTIONS_POLICY_RESULT=$(aws iam create-policy \
```

```
  --policy-name "StepFunctionsPolicy-${RANDOM_ID}" \
  --policy-document file://stepfunctions-policy.json 2>&1)

check_api_error "$STEPFUNCTIONS_POLICY_RESULT" "Create Step Functions policy"
echo "Step Functions policy created successfully"

# Get the policy ARN
STEPFUNCTIONS_POLICY_ARN=$(extract_json_field "$STEPFUNCTIONS_POLICY_RESULT"
 ".Policy.Arn")

if [ -z "$STEPFUNCTIONS_POLICY_ARN" ]; then
    handle_error "Failed to extract Step Functions policy ARN"
fi
echo "Step Functions policy ARN: $STEPFUNCTIONS_POLICY_ARN"

# Attach policy to the role
echo "Attaching Step Functions policy to role..."
ATTACH_RESULT=$(aws iam attach-role-policy \
  --role-name "$ROLE_NAME" \
  --policy-arn "$STEPFUNCTIONS_POLICY_ARN" 2>&1)

if [ $? -ne 0 ]; then
    handle_error "Failed to attach Step Functions policy to role: $ATTACH_RESULT"
fi

# Wait for role to propagate (IAM changes can take time to propagate)
wait_for_propagation "IAM role" 8

# Create state machine
echo "Creating state machine: $STATE_MACHINE_NAME"
SM_RESULT=$(aws stepfunctions create-state-machine \
  --name "$STATE_MACHINE_NAME" \
  --definition file://hello-world.json \
  --role-arn "$ROLE_ARN" \
  --type STANDARD \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-step-functions-gs
 2>&1)

check_api_error "$SM_RESULT" "Create state machine"
echo "State machine created successfully"

# Get the state machine ARN
STATE_MACHINE_ARN=$(extract_json_field "$SM_RESULT" ".stateMachineArn")
```

```
if [ -z "$STATE_MACHINE_ARN" ]; then
    handle_error "Failed to extract state machine ARN"
fi
echo "State machine ARN: $STATE_MACHINE_ARN"

# Step 2: Start the state machine execution
echo "Starting state machine execution..."
EXEC_RESULT=$(aws stepfunctions start-execution \
  --state-machine-arn "$STATE_MACHINE_ARN" \
  --name "hello001-${RANDOM_ID}" 2>&1)

check_api_error "$EXEC_RESULT" "Start execution"
echo "Execution started successfully"

# Get the execution ARN
EXECUTION_ARN=$(extract_json_field "$EXEC_RESULT" ".executionArn")

if [ -z "$EXECUTION_ARN" ]; then
    handle_error "Failed to extract execution ARN"
fi
echo "Execution ARN: $EXECUTION_ARN"

# Wait for execution to complete (the workflow has a 10-second wait state)
echo "Waiting for execution to complete (12 seconds)..."
sleep 12

# Check execution status
echo "Checking execution status..."
EXEC_STATUS=$(aws stepfunctions describe-execution \
  --execution-arn "$EXECUTION_ARN" 2>&1)

echo "Execution status: $EXEC_STATUS"

# Step 3: Update state machine to process external input
echo "Updating state machine to process external input..."
cat > updated-hello-world.json << 'EOF'
{
  "Comment": "A Hello World example of the Amazon States Language using a Pass
 state",
  "StartAt": "SetVariables",
  "States": {
    "SetVariables": {
      "Type": "Pass",
      "Parameters": {
```

```
        "IsHelloWorldExample.$": "$.hello_world",
        "ExecutionWaitTimeInSeconds.$": "$.wait"
      },
      "Next": "IsHelloWorldExample"
    },
    "IsHelloWorldExample": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": true,
          "Next": "WaitState"
        }
      ],
      "Default": "FailState"
    },
    "WaitState": {
      "Type": "Wait",
      "SecondsPath": "$.ExecutionWaitTimeInSeconds",
      "Next": "ParallelProcessing"
    },
    "ParallelProcessing": {
      "Type": "Parallel",
      "Branches": [
        {
          "StartAt": "Process1",
          "States": {
            "Process1": {
              "Type": "Pass",
              "Result": {
                "message": "Processing task 1"
              },
              "End": true
            }
          }
        },
        {
          "StartAt": "Process2",
          "States": {
            "Process2": {
              "Type": "Pass",
              "Result": {
                "message": "Processing task 2"
              },
```

```
              "End": true
            }
          }
        }
      ],
      "Next": "CheckpointState"
    },
    "CheckpointState": {
      "Type": "Pass",
      "Result": {
        "CheckpointMessage": "Workflow completed successfully!"
      },
      "Next": "SuccessState"
    },
    "SuccessState": {
      "Type": "Succeed"
    },
    "FailState": {
      "Type": "Fail",
      "Error": "NotHelloWorldExample",
      "Cause": "The IsHelloWorldExample value was false"
    }
  }
}
EOF

# Update state machine
echo "Updating state machine..."
UPDATE_RESULT=$(aws stepfunctions update-state-machine \
  --state-machine-arn "$STATE_MACHINE_ARN" \
  --definition file://updated-hello-world.json \
  --role-arn "$ROLE_ARN" 2>&1)

check_api_error "$UPDATE_RESULT" "Update state machine"
echo "State machine updated successfully"

# Create input file with strict validation
echo "Creating input file..."
cat > input.json << 'EOF'
{
  "wait": 5,
  "hello_world": true
}
EOF
```

```
# Validate input JSON
validate_json_file "input.json"

# Start execution with input
echo "Starting execution with input..."
EXEC2_RESULT=$(aws stepfunctions start-execution \
  --state-machine-arn "$STATE_MACHINE_ARN" \
  --name "hello002-${RANDOM_ID}" \
  --input file://input.json 2>&1)

check_api_error "$EXEC2_RESULT" "Start execution with input"
echo "Execution with input started successfully"

# Get the execution ARN
EXECUTION2_ARN=$(extract_json_field "$EXEC2_RESULT" ".executionArn")

if [ -z "$EXECUTION2_ARN" ]; then
    handle_error "Failed to extract execution ARN"
fi
echo "Execution ARN: $EXECUTION2_ARN"

# Wait for execution to complete (the workflow has a 5-second wait state)
echo "Waiting for execution to complete (8 seconds)..."
sleep 8

# Check execution status
echo "Checking execution status..."
EXEC2_STATUS=$(aws stepfunctions describe-execution \
  --execution-arn "$EXECUTION2_ARN" 2>&1)

echo "Execution status: $EXEC2_STATUS"

# Step 4: Integrate Amazon Comprehend for sentiment analysis (if available)
if [[ "$SKIP_COMPREHEND" == "false" ]]; then
    echo "Creating policy for Amazon Comprehend access with least privilege..."
    cat > comprehend-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "comprehend:DetectSentiment"
```

```
      ],
      "Resource": "*"
    }
  ]
}
EOF

    # Create policy
    echo "Creating IAM policy: $POLICY_NAME"
    POLICY_RESULT=$(aws iam create-policy \
      --policy-name "$POLICY_NAME" \
      --policy-document file://comprehend-policy.json 2>&1)

    check_api_error "$POLICY_RESULT" "Create Comprehend policy"
    echo "Comprehend policy created successfully"

    # Get policy ARN
    POLICY_ARN=$(extract_json_field "$POLICY_RESULT" ".Policy.Arn")

    if [ -z "$POLICY_ARN" ]; then
        handle_error "Failed to extract policy ARN"
    fi
    echo "Policy ARN: $POLICY_ARN"

    # Attach policy to role
    echo "Attaching policy to role..."
    ATTACH2_RESULT=$(aws iam attach-role-policy \
      --role-name "$ROLE_NAME" \
      --policy-arn "$POLICY_ARN" 2>&1)

    if [ $? -ne 0 ]; then
        handle_error "Failed to attach policy to role: $ATTACH2_RESULT"
    fi

    # Create updated state machine definition with sentiment analysis
    echo "Creating updated state machine definition with sentiment analysis..."
    cat > sentiment-hello-world.json << 'EOF'
{
  "Comment": "A Hello World example with sentiment analysis",
  "StartAt": "SetVariables",
  "States": {
    "SetVariables": {
      "Type": "Pass",
      "Parameters": {
```

```
        "IsHelloWorldExample.$": "$.hello_world",
        "ExecutionWaitTimeInSeconds.$": "$.wait",
        "FeedbackComment.$": "$.feedback_comment"
      },
      "Next": "IsHelloWorldExample"
    },
    "IsHelloWorldExample": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": true,
          "Next": "WaitState"
        }
      ],
      "Default": "DetectSentiment"
    },
    "WaitState": {
      "Type": "Wait",
      "SecondsPath": "$.ExecutionWaitTimeInSeconds",
      "Next": "ParallelProcessing"
    },
    "ParallelProcessing": {
      "Type": "Parallel",
      "Branches": [
        {
          "StartAt": "Process1",
          "States": {
            "Process1": {
              "Type": "Pass",
              "Result": {
                "message": "Processing task 1"
              },
              "End": true
            }
          }
        },
        {
          "StartAt": "Process2",
          "States": {
            "Process2": {
              "Type": "Pass",
              "Result": {
                "message": "Processing task 2"
```

```
              },
              "End": true
            }
          }
        }
      ],
      "Next": "CheckpointState"
    },
    "CheckpointState": {
      "Type": "Pass",
      "Result": {
        "CheckpointMessage": "Workflow completed successfully!"
      },
      "Next": "SuccessState"
    },
    "DetectSentiment": {
      "Type": "Task",
      "Resource": "arn:aws:states:::aws-sdk:comprehend:detectSentiment",
      "Parameters": {
        "LanguageCode": "en",
        "Text.$": "$.FeedbackComment"
      },
      "Next": "SuccessState"
    },
    "SuccessState": {
      "Type": "Succeed"
    }
  }
}
EOF

    # Validate sentiment state machine JSON
    validate_json_file "sentiment-hello-world.json"

    # Wait for IAM changes to propagate
    wait_for_propagation "IAM changes" 8

    # Update state machine
    echo "Updating state machine with sentiment analysis..."
    UPDATE2_RESULT=$(aws stepfunctions update-state-machine \
      --state-machine-arn "$STATE_MACHINE_ARN" \
      --definition file://sentiment-hello-world.json \
      --role-arn "$ROLE_ARN" 2>&1)
```

```
    check_api_error "$UPDATE2_RESULT" "Update state machine with sentiment
 analysis"
    echo "State machine updated with sentiment analysis successfully"

    # Create input file with feedback comment
    echo "Creating input file with feedback comment..."
    cat > sentiment-input.json << 'EOF'
{
  "hello_world": false,
  "wait": 5,
  "feedback_comment": "This getting started with Step Functions workshop is a
 challenge!"
}
EOF

    # Validate sentiment input JSON
    validate_json_file "sentiment-input.json"

    # Start execution with sentiment analysis input
    echo "Starting execution with sentiment analysis input..."
    EXEC3_RESULT=$(aws stepfunctions start-execution \
      --state-machine-arn "$STATE_MACHINE_ARN" \
      --name "hello003-${RANDOM_ID}" \
      --input file://sentiment-input.json 2>&1)

    check_api_error "$EXEC3_RESULT" "Start execution with sentiment analysis"
    echo "Execution with sentiment analysis started successfully"

    # Get the execution ARN
    EXECUTION3_ARN=$(extract_json_field "$EXEC3_RESULT" ".executionArn")

    if [ -z "$EXECUTION3_ARN" ]; then
        handle_error "Failed to extract execution ARN"
    fi
    echo "Execution ARN: $EXECUTION3_ARN"

    # Wait for execution to complete
    echo "Waiting for execution to complete (3 seconds)..."
    sleep 3

    # Check execution status
    echo "Checking execution status..."
    EXEC3_STATUS=$(aws stepfunctions describe-execution \
      --execution-arn "$EXECUTION3_ARN" 2>&1)
```

```
    echo "Execution status: $EXEC3_STATUS"
else
    echo "Skipping Amazon Comprehend integration (not available in
 $CURRENT_REGION)"
    EXECUTION3_ARN=""
fi

# Display summary of resources created
echo ""
echo "============================================="
echo "RESOURCES CREATED"
echo "============================================="
echo "State Machine: $STATE_MACHINE_ARN"
echo "IAM Role: $ROLE_NAME"
echo "Step Functions Policy: StepFunctionsPolicy-${RANDOM_ID}
 ($STEPFUNCTIONS_POLICY_ARN)"
if [[ "$SKIP_COMPREHEND" == "false" ]]; then
    echo "Comprehend Policy: $POLICY_NAME ($POLICY_ARN)"
fi
echo "Executions:"
echo "  - hello001-${RANDOM_ID}: $EXECUTION_ARN"
echo "  - hello002-${RANDOM_ID}: $EXECUTION2_ARN"
if [[ "$SKIP_COMPREHEND" == "false" ]]; then
    echo "  - hello003-${RANDOM_ID}: $EXECUTION3_ARN"
fi
echo "============================================="

# Cleanup
echo ""
echo "============================================="
echo "CLEANUP"
echo "============================================="
echo "Auto-cleanup enabled. Cleaning up resources..."

echo "All resources have been cleaned up."

echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreatePolicy

- CreateRole
- CreateStateMachine
- DeletePolicy
- DeleteRole
- DeleteStateMachine
- DescribeExecution
- DetachRolePolicy
- StartExecution
- UpdateStateMachine

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Manage IAM access keys using an AWS SDK

The following code example shows how to manage access keys.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create and list access keys.
- Find out when and how an access key was last used.
- Update and delete access keys.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM access key actions.

```
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

iam = boto3.resource("iam")

def list_keys(user_name):
    """
    Lists the keys owned by the specified user.

    :param user_name: The name of the user.
    :return: The list of keys owned by the user.
    """
    try:
        keys = list(iam.User(user_name).access_keys.all())
        logger.info("Got %s access keys for %s.", len(keys), user_name)
    except ClientError:
        logger.exception("Couldn't get access keys for %s.", user_name)
        raise
    else:
        return keys



def create_key(user_name):
    """
    Creates an access key for the specified user. Each user can have a
    maximum of two keys.
```

```
    :param user_name: The name of the user.
    :return: The created access key.
    """
    try:
        key_pair = iam.User(user_name).create_access_key_pair()
        logger.info(
            "Created access key pair for %s. Key ID is %s.",
            key_pair.user_name,
            key_pair.id,
        )
    except ClientError:
        logger.exception("Couldn't create access key pair for %s.", user_name)
        raise
    else:
        return key_pair



def get_last_use(key_id):
    """
    Gets information about when and how a key was last used.

    :param key_id: The ID of the key to look up.
    :return: Information about the key's last use.
    """
    try:
        response = iam.meta.client.get_access_key_last_used(AccessKeyId=key_id)
        last_used_date = response["AccessKeyLastUsed"].get("LastUsedDate", None)
        last_service = response["AccessKeyLastUsed"].get("ServiceName", None)
        logger.info(
            "Key %s was last used by %s on %s to access %s.",
            key_id,
            response["UserName"],
            last_used_date,
            last_service,
        )
    except ClientError:
        logger.exception("Couldn't get last use of key %s.", key_id)
        raise
    else:
        return response
```

```
def update_key(user_name, key_id, activate):
    """
    Updates the status of a key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to update.
    :param activate: When True, the key is activated. Otherwise, the key is
 deactivated.
    """

    try:
        key = iam.User(user_name).AccessKey(key_id)
        if activate:
            key.activate()
        else:
            key.deactivate()
        logger.info("%s key %s.", "Activated" if activate else "Deactivated",
 key_id)
    except ClientError:
        logger.exception(
            "Couldn't %s key %s.", "Activate" if activate else "Deactivate",
 key_id
        )
        raise



def delete_key(user_name, key_id):
    """
    Deletes a user's access key.

    :param user_name: The user that owns the key.
    :param key_id: The ID of the key to delete.
    """

    try:
        key = iam.AccessKey(user_name, key_id)
        key.delete()
        logger.info("Deleted access key %s for %s.", key.id, key.user_name)
    except ClientError:
        logger.exception("Couldn't delete key %s for %s", key_id, user_name)
        raise
```

Use the wrapper functions to perform access key actions for the current user.

```
def usage_demo():
    """Shows how to create and manage access keys."""

    def print_keys():
        """Gets and prints the current keys for a user."""
        current_keys = list_keys(current_user_name)
        print("The current user's keys are now:")
        print(*[f"{key.id}: {key.status}" for key in current_keys], sep="\n")

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    print("-" * 88)
    print("Welcome to the AWS Identity and Account Management access key demo.")
    print("-" * 88)
    current_user_name = iam.CurrentUser().user_name
    print(
        f"This demo creates an access key for the current user "
        f"({current_user_name}), manipulates the key in a few ways, and then "
        f"deletes it."
    )
    all_keys = list_keys(current_user_name)
    if len(all_keys) == 2:
        print(
            "The current user already has the maximum of 2 access keys. To run "
            "this demo, either delete one of the access keys or use a user "
            "that has only 1 access key."
        )
    else:
        new_key = create_key(current_user_name)
        print(f"Created a new key with id {new_key.id} and secret
 {new_key.secret}.")
        print_keys()
        existing_key = next(key for key in all_keys if key != new_key)
        last_use = get_last_use(existing_key.id)["AccessKeyLastUsed"]
        print(
            f"Key {all_keys[0].id} was last used to access
 {last_use['ServiceName']} "
            f"on {last_use['LastUsedDate']}"
        )
        update_key(current_user_name, new_key.id, False)
```

```
        print(f"Key {new_key.id} is now deactivated.")
        print_keys()
        delete_key(current_user_name, new_key.id)
        print_keys()
        print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateAccessKey
  - DeleteAccessKey
  - GetAccessKeyLastUsed
  - ListAccessKeys
  - UpdateAccessKey

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Manage IAM policies using an AWS SDK

The following code example shows how to:

- Create and list policies.
- Create and get policy versions.
- Roll back a policy to a previous version.
- Delete policies.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM policy actions.

```
import json
import logging
import operator
import pprint
import time

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource("iam")

def create_policy(name, description, actions, resource_arn):
    """
    Creates a policy that contains a single statement.

    :param name: The name of the policy to create.
    :param description: The description of the policy.
    :param actions: The actions allowed by the policy. These typically take the
                    form of service:action, such as s3:PutObject.
    :param resource_arn: The Amazon Resource Name (ARN) of the resource this
 policy
                         applies to. This ARN can contain wildcards, such as
                         'arn:aws:s3:::amzn-s3-demo-bucket/*' to allow actions on
 all objects
                         in the bucket named 'amzn-s3-demo-bucket'.
    :return: The newly created policy.
    """
    policy_doc = {
        "Version":"2012-10-17",
        "Statement": [{"Effect": "Allow", "Action": actions, "Resource":
 resource_arn}],
    }
    try:
        policy = iam.create_policy(
            PolicyName=name,
            Description=description,
            PolicyDocument=json.dumps(policy_doc),
        )
        logger.info("Created policy %s.", policy.arn)
    except ClientError:
        logger.exception("Couldn't create policy %s.", name)
```

```
        raise
    else:
        return policy



def list_policies(scope):
    """
    Lists the policies in the current account.

    :param scope: Limits the kinds of policies that are returned. For example,
                  'Local' specifies that only locally managed policies are
 returned.
    :return: The list of policies.
    """
    try:
        policies = list(iam.policies.filter(Scope=scope))
        logger.info("Got %s policies in scope '%s'.", len(policies), scope)
    except ClientError:
        logger.exception("Couldn't get policies for scope '%s'.", scope)
        raise
    else:
        return policies



def create_policy_version(policy_arn, actions, resource_arn, set_as_default):
    """
    Creates a policy version. Policies can have up to five versions. The default
    version is the one that is used for all resources that reference the policy.

    :param policy_arn: The ARN of the policy.
    :param actions: The actions to allow in the policy version.
    :param resource_arn: The ARN of the resource this policy version applies to.
    :param set_as_default: When True, this policy version is set as the default
                           version for the policy. Otherwise, the default
                           is not changed.
    :return: The newly created policy version.
    """
    policy_doc = {
        "Version":"2012-10-17",
        "Statement": [{"Effect": "Allow", "Action": actions, "Resource":
 resource_arn}],
    }
```

```
    try:
        policy = iam.Policy(policy_arn)
        policy_version = policy.create_version(
            PolicyDocument=json.dumps(policy_doc), SetAsDefault=set_as_default
        )
        logger.info(
            "Created policy version %s for policy %s.",
            policy_version.version_id,
            policy_version.arn,
        )
    except ClientError:
        logger.exception("Couldn't create a policy version for %s.", policy_arn)
        raise
    else:
        return policy_version



def get_default_policy_statement(policy_arn):
    """
    Gets the statement of the default version of the specified policy.

    :param policy_arn: The ARN of the policy to look up.
    :return: The statement of the default policy version.
    """
    try:
        policy = iam.Policy(policy_arn)
        # To get an attribute of a policy, the SDK first calls get_policy.
        policy_doc = policy.default_version.document
        policy_statement = policy_doc.get("Statement", None)
        logger.info("Got default policy doc for %s.", policy.policy_name)
        logger.info(policy_doc)
    except ClientError:
        logger.exception("Couldn't get default policy statement for %s.",
 policy_arn)
        raise
    else:
        return policy_statement



def rollback_policy_version(policy_arn):
    """
    Rolls back to the previous default policy, if it exists.
```

```
    1. Gets the list of policy versions in order by date.
    2. Finds the default.
    3. Makes the previous policy the default.
    4. Deletes the old default version.

    :param policy_arn: The ARN of the policy to roll back.
    :return: The default version of the policy after the rollback.
    """
    try:
        policy_versions = sorted(
            iam.Policy(policy_arn).versions.all(),
            key=operator.attrgetter("create_date"),
        )
        logger.info("Got %s versions for %s.", len(policy_versions), policy_arn)
    except ClientError:
        logger.exception("Couldn't get versions for %s.", policy_arn)
        raise

    default_version = None
    rollback_version = None
    try:
        while default_version is None:
            ver = policy_versions.pop()
            if ver.is_default_version:
                default_version = ver
        rollback_version = policy_versions.pop()
        rollback_version.set_as_default()
        logger.info("Set %s as the default version.",
rollback_version.version_id)
        default_version.delete()
        logger.info("Deleted original default version %s.",
default_version.version_id)
    except IndexError:
        if default_version is None:
            logger.warning("No default version found for %s.", policy_arn)
        elif rollback_version is None:
            logger.warning(
                "Default version %s found for %s, but no previous version exists,
so "
                "nothing to roll back to.",
                default_version.version_id,
                policy_arn,
            )
```

```
    except ClientError:
        logger.exception("Couldn't roll back version for %s.", policy_arn)
        raise
    else:
        return rollback_version



def delete_policy(policy_arn):
    """
    Deletes a policy.

    :param policy_arn: The ARN of the policy to delete.
    """
    try:
        iam.Policy(policy_arn).delete()
        logger.info("Deleted policy %s.", policy_arn)
    except ClientError:
        logger.exception("Couldn't delete policy %s.", policy_arn)
        raise
```

Use the wrapper functions to create policies, update versions, and get information about them.

```
def usage_demo():
    """Shows how to use the policy functions."""
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    print("-" * 88)
    print("Welcome to the AWS Identity and Account Management policy demo.")
    print("-" * 88)
    print(
        "Policies let you define sets of permissions that can be attached to "
        "other IAM resources, like users and roles."
    )
    bucket_arn = f"arn:aws:s3:::amzn-s3-demo-bucket"
    policy = create_policy(
        "demo-iam-policy",
        "Policy for IAM demonstration.",
        ["s3:ListObjects"],
        bucket_arn,
```

```
    )
    print(f"Created policy {policy.policy_name}.")
    policies = list_policies("Local")
    print(f"Your account has {len(policies)} managed policies:")
    print(*[pol.policy_name for pol in policies], sep=", ")
    time.sleep(1)
    policy_version = create_policy_version(
        policy.arn, ["s3:PutObject"], bucket_arn, True
    )
    print(
        f"Added policy version {policy_version.version_id} to policy "
        f"{policy.policy_name}."
    )
    default_statement = get_default_policy_statement(policy.arn)
    print(f"The default policy statement for {policy.policy_name} is:")
    pprint.pprint(default_statement)
    rollback_version = rollback_policy_version(policy.arn)
    print(
        f"Rolled back to version {rollback_version.version_id} for "
        f"{policy.policy_name}."
    )
    default_statement = get_default_policy_statement(policy.arn)
    print(f"The default policy statement for {policy.policy_name} is now:")
    pprint.pprint(default_statement)
    delete_policy(policy.arn)
    print(f"Deleted policy {policy.policy_name}.")
    print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreatePolicy
  - CreatePolicyVersion
  - DeletePolicy
  - DeletePolicyVersion
  - GetPolicyVersion
  - ListPolicies
  - ListPolicyVersions
  - SetDefaultPolicyVersion

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Manage IAM roles using an AWS SDK

The following code example shows how to:

- Create an IAM role.
- Attach and detach policies for a role.
- Delete a role.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM role actions.

```
import json
import logging
import pprint

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource("iam")

def create_role(role_name, allowed_services):
    """
    Creates a role that lets a list of specified services assume the role.

    :param role_name: The name of the role.
    :param allowed_services: The services that can assume the role.
    :return: The newly created role.
```

```
    """
    trust_policy = {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {"Service": service},
                "Action": "sts:AssumeRole",
            }
            for service in allowed_services
        ],
    }

    try:
        role = iam.create_role(
            RoleName=role_name, AssumeRolePolicyDocument=json.dumps(trust_policy)
        )
        logger.info("Created role %s.", role.name)
    except ClientError:
        logger.exception("Couldn't create role %s.", role_name)
        raise
    else:
        return role



def attach_policy(role_name, policy_arn):
    """
    Attaches a policy to a role.

    :param role_name: The name of the role. **Note** this is the name, not the
 ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Role(role_name).attach_policy(PolicyArn=policy_arn)
        logger.info("Attached policy %s to role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception("Couldn't attach policy %s to role %s.", policy_arn,
 role_name)
        raise
```

```
def detach_policy(role_name, policy_arn):
    """
    Detaches a policy from a role.

    :param role_name: The name of the role. **Note** this is the name, not the
 ARN.
    :param policy_arn: The ARN of the policy.
    """
    try:
        iam.Role(role_name).detach_policy(PolicyArn=policy_arn)
        logger.info("Detached policy %s from role %s.", policy_arn, role_name)
    except ClientError:
        logger.exception(
            "Couldn't detach policy %s from role %s.", policy_arn, role_name
        )
        raise



def delete_role(role_name):
    """
    Deletes a role.

    :param role_name: The name of the role to delete.
    """
    try:
        iam.Role(role_name).delete()
        logger.info("Deleted role %s.", role_name)
    except ClientError:
        logger.exception("Couldn't delete role %s.", role_name)
        raise
```

Use the wrapper functions to create a role, then attach and detach a policy.

```
def usage_demo():
    """Shows how to use the role functions."""
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    print("-" * 88)
    print("Welcome to the AWS Identity and Account Management role demo.")
    print("-" * 88)
```

```
    print(
        "Roles let you define sets of permissions and can be assumed by "
        "other entities, like users and services."
    )
    print("The first 10 roles currently in your account are:")
    roles = list_roles(10)
    print(f"The inline policies for role {roles[0].name} are:")
    list_policies(roles[0].name)
    role = create_role(
        "demo-iam-role", ["lambda.amazonaws.com",
"batchoperations.s3.amazonaws.com"]
    )
    print(f"Created role {role.name}, with trust policy:")
    pprint.pprint(role.assume_role_policy_document)
    policy_arn = "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
    attach_policy(role.name, policy_arn)
    print(f"Attached policy {policy_arn} to {role.name}.")
    print(f"Policies attached to role {role.name} are:")
    list_attached_policies(role.name)
    detach_policy(role.name, policy_arn)
    print(f"Detached policy {policy_arn} from {role.name}.")
    delete_role(role.name)
    print(f"Deleted {role.name}.")
    print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - AttachRolePolicy
  - CreateRole
  - DeleteRole
  - DetachRolePolicy

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Manage your IAM account using an AWS SDK

The following code example shows how to:

- Get and update the account alias.
- Generate a report of users and credentials.
- Get a summary of account usage.
- Get details for all users, groups, roles, and policies in your account, including their relationships to each other.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions that wrap IAM account actions.

```
import logging
import pprint
import sys
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
iam = boto3.resource("iam")

def list_aliases():
    """
    Gets the list of aliases for the current account. An account has at most one
 alias.

    :return: The list of aliases for the account.
    """
    try:
        response = iam.meta.client.list_account_aliases()
        aliases = response["AccountAliases"]
        if len(aliases) > 0:
            logger.info("Got aliases for your account: %s.", ",".join(aliases))
```

```
        else:
            logger.info("Got no aliases for your account.")
    except ClientError:
        logger.exception("Couldn't list aliases for your account.")
        raise
    else:
        return response["AccountAliases"]



def create_alias(alias):
    """
    Creates an alias for the current account. The alias can be used in place of
 the
    account ID in the sign-in URL. An account can have only one alias. When a new
    alias is created, it replaces any existing alias.

    :param alias: The alias to assign to the account.
    """

    try:
        iam.create_account_alias(AccountAlias=alias)
        logger.info("Created an alias '%s' for your account.", alias)
    except ClientError:
        logger.exception("Couldn't create alias '%s' for your account.", alias)
        raise



def delete_alias(alias):
    """
    Removes the alias from the current account.

    :param alias: The alias to remove.
    """
    try:
        iam.meta.client.delete_account_alias(AccountAlias=alias)
        logger.info("Removed alias '%s' from your account.", alias)
    except ClientError:
        logger.exception("Couldn't remove alias '%s' from your account.", alias)
        raise
```

```
def generate_credential_report():
    """
    Starts generation of a credentials report about the current account. After
    calling this function to generate the report, call get_credential_report
    to get the latest report. A new report can be generated a minimum of four
 hours
    after the last one was generated.
    """
    try:
        response = iam.meta.client.generate_credential_report()
        logger.info(
            "Generating credentials report for your account. " "Current state is
 %s.",
            response["State"],
        )
    except ClientError:
        logger.exception("Couldn't generate a credentials report for your
 account.")
        raise
    else:
        return response



def get_credential_report():
    """
    Gets the most recently generated credentials report about the current
 account.

    :return: The credentials report.
    """
    try:
        response = iam.meta.client.get_credential_report()
        logger.debug(response["Content"])
    except ClientError:
        logger.exception("Couldn't get credentials report.")
        raise
    else:
        return response["Content"]



def get_summary():
    """
```

```
    Gets a summary of account usage.

    :return: The summary of account usage.
    """
    try:
        summary = iam.AccountSummary()
        logger.debug(summary.summary_map)
    except ClientError:
        logger.exception("Couldn't get a summary for your account.")
        raise
    else:
        return summary.summary_map



def get_authorization_details(response_filter):
    """
    Gets an authorization detail report for the current account.

    :param response_filter: A list of resource types to include in the report,
 such
                            as users or roles. When not specified, all resources
                            are included.
    :return: The authorization detail report.
    """
    try:
        account_details = iam.meta.client.get_account_authorization_details(
            Filter=response_filter
        )
        logger.debug(account_details)
    except ClientError:
        logger.exception("Couldn't get details for your account.")
        raise
    else:
        return account_details
```

Call wrapper functions to change the account alias and to get reports about the account.

```
def usage_demo():
    """Shows how to use the account functions."""
```

```
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    print("-" * 88)
    print("Welcome to the AWS Identity and Account Management account demo.")
    print("-" * 88)
    print(
        "Setting an account alias lets you use the alias in your sign-in URL "
        "instead of your account number."
    )
    old_aliases = list_aliases()
    if len(old_aliases) > 0:
        print(f"Your account currently uses '{old_aliases[0]}' as its alias.")
    else:
        print("Your account currently has no alias.")
    for index in range(1, 3):
        new_alias = f"alias-{index}-{time.time_ns()}"
        print(f"Setting your account alias to {new_alias}")
        create_alias(new_alias)
    current_aliases = list_aliases()
    print(f"Your account alias is now {current_aliases}.")
    delete_alias(current_aliases[0])
    print(f"Your account now has no alias.")
    if len(old_aliases) > 0:
        print(f"Restoring your original alias back to {old_aliases[0]}...")
        create_alias(old_aliases[0])

    print("-" * 88)
    print("You can get various reports about your account.")
    print("Let's generate a credentials report...")
    report_state = None
    while report_state != "COMPLETE":
        cred_report_response = generate_credential_report()
        old_report_state = report_state
        report_state = cred_report_response["State"]
        if report_state != old_report_state:
            print(report_state, sep="")
        else:
            print(".", sep="")
        sys.stdout.flush()
        time.sleep(1)
    print()
    cred_report = get_credential_report()
    col_count = 3
    print(f"Got credentials report. Showing only the first {col_count} columns.")
    cred_lines = [
```

```
        line.split(",")[:col_count] for line in
 cred_report.decode("utf-8").split("\n")
    ]
    col_width = max([len(item) for line in cred_lines for item in line]) + 2
    for line in cred_report.decode("utf-8").split("\n"):
        print(
            "".join(element.ljust(col_width) for element in line.split(",")
[:col_count])
        )

    print("-" * 88)
    print("Let's get an account summary.")
    summary = get_summary()
    print("Here's your summary:")
    pprint.pprint(summary)

    print("-" * 88)
    print("Let's get authorization details!")
    details = get_authorization_details([])
    see_details = input("These are pretty long, do you want to see them (y/n)? ")
    if see_details.lower() == "y":
        pprint.pprint(details)

    print("-" * 88)
    pw_policy_created = None
    see_pw_policy = input("Want to see the password policy for the account (y/n)?
 ")
    if see_pw_policy.lower() == "y":
        while True:
            if print_password_policy():
                break
            else:
                answer = input(
                    "Do you want to create a default password policy (y/n)? "
                )
                if answer.lower() == "y":
                    pw_policy_created = iam.create_account_password_policy()
                else:
                    break
    if pw_policy_created is not None:
        answer = input("Do you want to delete the password policy (y/n)? ")
        if answer.lower() == "y":
            pw_policy_created.delete()
            print("Password policy deleted.")
```

```
    print("The SAML providers for your account are:")
    list_saml_providers(10)

    print("-" * 88)
    print("Thanks for watching.")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateAccountAlias
  - DeleteAccountAlias
  - GenerateCredentialReport
  - GetAccountAuthorizationDetails
  - GetAccountSummary
  - GetCredentialReport
  - ListAccountAliases

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Moving hardcoded secrets to secure secret storage

The following code example shows how to:

- Create IAM roles
- Create a secret in Secrets Manager
- Update your application code
- Update the secret
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Script to move hardcoded secrets to AWS Secrets Manager
# This script demonstrates how to create IAM roles, store a secret in AWS Secrets
 Manager,
# and set up appropriate permissions

set -euo pipefail

# Set up logging
LOG_FILE="secrets_manager_tutorial.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting AWS Secrets Manager tutorial script at $(date)"
echo "================================================="

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2

    if echo "$output" | grep -qi "error"; then
        echo "ERROR: Command failed: $cmd"
        echo "$output"
        cleanup_resources
        exit 1
    fi
}

# Function to generate a random identifier using secure method
generate_random_id() {
    python3 -c "import secrets; print('sm' + secrets.token_hex(4))"
```

```
}

# Function to safely clean up resources
cleanup_resources() {
    echo ""
    echo "========================================="
    echo "RESOURCES CREATED"
    echo "========================================="

    if [ -n "${SECRET_NAME:-}" ]; then
        echo "Secret: $SECRET_NAME"
    fi

    if [ -n "${RUNTIME_ROLE_NAME:-}" ]; then
        echo "IAM Role: $RUNTIME_ROLE_NAME"
    fi

    if [ -n "${ADMIN_ROLE_NAME:-}" ]; then
        echo "IAM Role: $ADMIN_ROLE_NAME"
    fi

    echo ""
    echo "========================================="
    echo "CLEANUP CONFIRMATION"
    echo "========================================="
    echo "Cleaning up all created resources..."

    # Delete secret if it exists
    if [ -n "${SECRET_NAME:-}" ]; then
        echo "Deleting secret: $SECRET_NAME"
        aws secretsmanager delete-secret --secret-id "$SECRET_NAME" --force-
delete-without-recovery 2>/dev/null || true
    fi

    # Detach policies and delete runtime role if it exists
    if [ -n "${RUNTIME_ROLE_NAME:-}" ]; then
        echo "Deleting inline policies from runtime role: $RUNTIME_ROLE_NAME"
        for policy in $(aws iam list-role-policies --role-name
 "$RUNTIME_ROLE_NAME" --query 'PolicyNames[]' --output text 2>/dev/null || true);
 do
            aws iam delete-role-policy --role-name "$RUNTIME_ROLE_NAME" --policy-
name "$policy" 2>/dev/null || true
        done
        echo "Deleting IAM role: $RUNTIME_ROLE_NAME"
```

```
        aws iam delete-role --role-name "$RUNTIME_ROLE_NAME" 2>/dev/null || true
    fi

    # Detach policies and delete admin role if it exists
    if [ -n "${ADMIN_ROLE_NAME:-}" ]; then
        echo "Detaching policy from role: $ADMIN_ROLE_NAME"
        aws iam detach-role-policy --role-name "$ADMIN_ROLE_NAME" --policy-arn
 "arn:aws:iam::aws:policy/SecretsManagerReadWrite" 2>/dev/null || true

        for policy in $(aws iam list-role-policies --role-name "$ADMIN_ROLE_NAME"
 --query 'PolicyNames[]' --output text 2>/dev/null || true); do
            aws iam delete-role-policy --role-name "$ADMIN_ROLE_NAME" --policy-
name "$policy" 2>/dev/null || true
        done

        echo "Deleting IAM role: $ADMIN_ROLE_NAME"
        aws iam delete-role --role-name "$ADMIN_ROLE_NAME" 2>/dev/null || true
    fi

    echo "Cleanup completed."
}

# Trap to ensure cleanup on script exit
trap 'echo "Script interrupted. Running cleanup..."; cleanup_resources' INT TERM
 EXIT

# Generate random identifiers for resources
ADMIN_ROLE_NAME="SecretsManagerAdmin-$(generate_random_id)"
RUNTIME_ROLE_NAME="RoleToRetrieveSecretAtRuntime-$(generate_random_id)"
SECRET_NAME="MyAPIKey-$(generate_random_id)"

echo "Using the following resource names:"
echo "Admin Role: $ADMIN_ROLE_NAME"
echo "Runtime Role: $RUNTIME_ROLE_NAME"
echo "Secret Name: $SECRET_NAME"
echo ""

# Step 1: Create IAM roles
echo "Creating IAM roles..."

# Create assume role policy document
ASSUME_ROLE_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF
)

# Create the SecretsManagerAdmin role
echo "Creating admin role: $ADMIN_ROLE_NAME"
ADMIN_ROLE_OUTPUT=$(aws iam create-role \
    --role-name "$ADMIN_ROLE_NAME" \
    --assume-role-policy-document "$ASSUME_ROLE_POLICY" 2>&1)

check_error "$ADMIN_ROLE_OUTPUT" "create-role for admin"
echo "$ADMIN_ROLE_OUTPUT"
aws iam tag-role --role-name "$ADMIN_ROLE_NAME" --tags Key=project,Value=doc-
smith Key=tutorial,Value=aws-secrets-manager-gs

# Attach the SecretsManagerReadWrite policy to the admin role
echo "Attaching SecretsManagerReadWrite policy to admin role"
ATTACH_POLICY_OUTPUT=$(aws iam attach-role-policy \
    --role-name "$ADMIN_ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/SecretsManagerReadWrite" 2>&1)

check_error "$ATTACH_POLICY_OUTPUT" "attach-role-policy for admin"
echo "Policy attached successfully"

# Create the RoleToRetrieveSecretAtRuntime role
echo "Creating runtime role: $RUNTIME_ROLE_NAME"
RUNTIME_ROLE_OUTPUT=$(aws iam create-role \
    --role-name "$RUNTIME_ROLE_NAME" \
    --assume-role-policy-document "$ASSUME_ROLE_POLICY" 2>&1)

check_error "$RUNTIME_ROLE_OUTPUT" "create-role for runtime"
echo "$RUNTIME_ROLE_OUTPUT"
aws iam tag-role --role-name "$RUNTIME_ROLE_NAME" --tags Key=project,Value=doc-
smith Key=tutorial,Value=aws-secrets-manager-gs
```

```
# Wait for roles to be fully created
echo "Waiting for IAM roles to be fully created..."
sleep 10

# Step 2: Create a secret in AWS Secrets Manager
echo "Creating secret in AWS Secrets Manager..."

# Generate secure secret value using environment variable or secure method
# WARNING: In production, use secure methods to inject secrets (AWS CodeBuild,
 parameter store, etc.)
if [ -z "${TUTORIAL_SECRET_VALUE:-}" ]; then
    SECRET_VALUE=$(python3 -c "import json;
 print(json.dumps({'ClientID':'my_client_id','ClientSecret':__import__('secrets').token_u
else
    SECRET_VALUE="$TUTORIAL_SECRET_VALUE"
fi

CREATE_SECRET_OUTPUT=$(aws secretsmanager create-secret \
    --name "$SECRET_NAME" \
    --description "API key for my application" \
    --secret-string "$SECRET_VALUE" \
    --add-replica-regions 'Region=us-east-1' \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-secrets-manager-gs
 2>&1)

check_error "$CREATE_SECRET_OUTPUT" "create-secret"
echo "$CREATE_SECRET_OUTPUT"

# Get AWS account ID
echo "Getting AWS account ID..."
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text 2>&1)
check_error "$ACCOUNT_ID" "get-caller-identity"
echo "Account ID: $ACCOUNT_ID"

# Get secret ARN for precise resource policy
echo "Getting secret ARN..."
SECRET_ARN=$(aws secretsmanager describe-secret \
    --secret-id "$SECRET_NAME" \
    --query 'ARN' \
    --output text 2>&1)
check_error "$SECRET_ARN" "describe-secret"

# Add resource policy to the secret with least privilege
echo "Adding resource policy to secret..."
```

```
RESOURCE_POLICY=$(cat <<EOF
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRuntimeRoleReadSecret",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::$ACCOUNT_ID:role/$RUNTIME_ROLE_NAME"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "$SECRET_ARN",
            "Condition": {
                "StringEquals": {
                    "secretsmanager:VersionStage": "AWSCURRENT"
                }
            }
        }
    ]
}
EOF
)

PUT_POLICY_OUTPUT=$(aws secretsmanager put-resource-policy \
    --secret-id "$SECRET_NAME" \
    --resource-policy "$RESOURCE_POLICY" \
    --block-public-policy 2>&1)

check_error "$PUT_POLICY_OUTPUT" "put-resource-policy"
echo "Resource policy added successfully"

# Enable rotation policy recommendation
echo "Enabling secret metadata tags for rotation tracking..."
aws secretsmanager tag-resource \
    --secret-id "$SECRET_NAME" \
    --tags Key=Purpose,Value=Tutorial Key=AutoRotation,Value=Recommended 2>/dev/
null || true

# Step 3: Demonstrate retrieving the secret
echo "Retrieving the secret value (for demonstration purposes)..."
GET_SECRET_OUTPUT=$(aws secretsmanager get-secret-value \
    --secret-id "$SECRET_NAME" 2>&1)

check_error "$GET_SECRET_OUTPUT" "get-secret-value"
```

```
echo "Secret retrieved successfully. Secret metadata:"
echo "$GET_SECRET_OUTPUT" | jq '{ARN: .ARN, Name: .Name,
 LastUpdatedDate: .LastUpdatedDate, VersionIdsToStages: .VersionIdsToStages}' 2>/
dev/null || echo "Secret metadata retrieved (jq not available)"

# Step 4: Update the secret with new values
echo "Updating the secret with new values..."
UPDATE_SECRET_VALUE=$(python3 -c "import json;
 print(json.dumps({'ClientID':'my_new_client_id','ClientSecret':__import__('secrets').tok

UPDATE_SECRET_OUTPUT=$(aws secretsmanager update-secret \
    --secret-id "$SECRET_NAME" \
    --secret-string "$UPDATE_SECRET_VALUE" 2>&1)

check_error "$UPDATE_SECRET_OUTPUT" "update-secret"
echo "Secret updated successfully"

# Step 5: Verify the updated secret
echo "Verifying the updated secret..."
VERIFY_SECRET_OUTPUT=$(aws secretsmanager get-secret-value \
    --secret-id "$SECRET_NAME" 2>&1)

check_error "$VERIFY_SECRET_OUTPUT" "get-secret-value for verification"
echo "Updated secret retrieved successfully. Secret metadata:"
echo "$VERIFY_SECRET_OUTPUT" | jq '{ARN: .ARN, Name: .Name,
 LastUpdatedDate: .LastUpdatedDate, VersionIdsToStages: .VersionIdsToStages}' 2>/
dev/null || echo "Secret metadata retrieved (jq not available)"

# Step 6: Display rotation recommendations
echo ""
echo "Rotation Configuration Recommendations:"
echo "======================================="
DESCRIBE_OUTPUT=$(aws secretsmanager describe-secret --secret-id "$SECRET_NAME"
 2>&1)
if echo "$DESCRIBE_OUTPUT" | grep -q "RotationRules"; then
    echo "Current rotation configuration:"
    echo "$DESCRIBE_OUTPUT" | jq '.RotationRules' 2>/dev/null || echo "Rotation
 rules available"
else
    echo "No automatic rotation configured. Consider enabling rotation with:"
    echo "aws secretsmanager rotate-secret --secret-id $SECRET_NAME --rotation-
lambda-arn arn:aws:lambda:REGION:ACCOUNT:function:FUNCTION_NAME --rotation-rules
 AutomaticallyAfterDays=30"
fi
```

```
echo ""
echo "============================================================"
echo "Tutorial completed successfully!"
echo ""
echo "Summary of what we did:"
echo "1. Created IAM roles for managing and retrieving secrets"
echo "2. Created a secret in AWS Secrets Manager with secure generation"
echo "3. Added a least-privilege resource policy to control access to the secret"
echo "4. Retrieved the secret value (simulating application access)"
echo "5. Updated the secret with cryptographically secure values"
echo "6. Verified the updated secret"
echo ""
echo "Security best practices applied:"
echo "- Used cryptographically secure random ID generation"
echo "- Applied least-privilege resource policies with version stages"
echo "- Tagged resources for rotation tracking"
echo "- Blocked public access to secrets"
echo "- Used ARN-specific permissions instead of wildcards"
echo ""
echo "Next steps you might want to consider:"
echo "- Enable automatic secret rotation with AWS Lambda"
echo "- Implement secret caching in your application"
echo "- Set up CloudTrail logging for secret access auditing"
echo "- Use AWS CodeGuru Reviewer to find hardcoded secrets in your code"
echo "- For multi-region applications, replicate your secrets across regions"
echo "- Configure VPC endpoints for private access to Secrets Manager"
echo ""

echo "Script completed at $(date)"
exit 0
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateRole
  - CreateSecret
  - DeleteRole
  - DeleteSecret
  - DetachRolePolicy
  - GetCallerIdentity

- GetSecretValue
- PutResourcePolicy
- UpdateSecret

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Allows the AWS Compute Optimizer Automation feature to apply recommended actions

The following code example shows how to This permission-based policy allows the AWS Compute Optimizer Automation feature to apply recommended actions

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "aco-automation.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Policy to enable Automation across your organization

The following code example shows how to This permission-based policy enables Automation across your organization

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/aco-
automation.amazonaws.com/AWSServiceRoleForComputeOptimizerAutomation",
            "Condition": {"StringLike": {"iam:AWSServiceName": "aco-
automation.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PutRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/aco-
automation.amazonaws.com/AWSServiceRoleForComputeOptimizerAutomation"
        },
        {
            "Effect": "Allow",
            "Action": "aco-automation:UpdateEnrollmentConfiguration",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "aco-automation:AssociateAccounts",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "aco-automation:DisassociateAccounts",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "aco-automation:ListAccounts",
            "Resource": "*"
        }
    ]
```

```
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Policy to enable Automation for your account

The following code example shows how to This permission-based policy enablesAutomation for your account

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/aco-
automation.amazonaws.com/AWSServiceRoleForComputeOptimizerAutomation",
            "Condition": {"StringLike": {"iam:AWSServiceName": "aco-
automation.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PutRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/aco-
automation.amazonaws.com/AWSServiceRoleForComputeOptimizerAutomation"
        },
        {
            "Effect": "Allow",
            "Action": "aco-automation:UpdateEnrollmentConfiguration",
            "Resource": "*"
        }
    ]
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Policy to grant full access to Compute Optimizer Automation for a management account of an organization

The following code example shows how to This permission-based policy grants full access to Compute Optimizer Automation for a management account of an organization

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "aco-automation:*",
               "ec2:DescribeVolumes",
               "organizations:ListAccounts",
               "organizations:DescribeOrganization",
               "organizations:DescribeAccount",
               "organizations:EnableAWSServiceAccess",
               "organizations:ListDelegatedAdministrators",
               "organizations:RegisterDelegatedAdministrator",
               "organizations:DeregisterDelegatedAdministrator"
            ],
            "Resource": "*"
        }
    ]
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Policy to grant full access to Compute Optimizer Automation for standalone AWS accounts

The following code example shows how to This permission-based policy grant full access to Compute Optimizer Automation for standalone AWS accounts

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "aco-automation:*",
            "ec2:DescribeVolumes"
            ],
            "Resource": "*"
        }
    ]
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Policy to grant read-only access to Compute Optimizer Automation for a management account of an organization

The following code example shows how to This permission-based policy grants read-only access to Compute Optimizer Automation for a management account of an organization

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
               "aco-automation:GetEnrollmentConfiguration",
               "aco-automation:GetAutomationEvent",
               "aco-automation:GetAutomationRule",
               "aco-automation:ListAccounts",
               "aco-automation:ListAutomationEvents",
               "aco-automation:ListAutomationEventSteps",
               "aco-automation:ListAutomationEventSummaries",
               "aco-automation:ListAutomationRules",
               "aco-automation:ListAutomationRulePreview",
               "aco-automation:ListAutomationRulePreviewSummaries",
               "aco-automation:ListRecommendedActions",
               "aco-automation:ListRecommendedActionSummaries",
               "aco-automation:ListTagsForResource",
               "ec2:DescribeVolumes"
            ],
            "Resource": "*"
        }
    ]
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Policy to grant read-only access to Compute Optimizer Automation for standalone AWS accounts

The following code example shows how to This permission-based policy grants read-only access to Compute Optimizer Automation for standalone AWS accounts

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "aco-automation:GetEnrollmentConfiguration",
               "aco-automation:GetAutomationEvent",
```

```
                "aco-automation:GetAutomationRule",
                "aco-automation:ListAutomationEvents",
                "aco-automation:ListAutomationEventSteps",
                "aco-automation:ListAutomationEventSummaries",
                "aco-automation:ListAutomationRules",
                "aco-automation:ListAutomationRulePreview",
                "aco-automation:ListAutomationRulePreviewSummaries",
                "aco-automation:ListRecommendedActions",
                "aco-automation:ListRecommendedActionSummaries",
                "aco-automation:ListTagsForResource",
                "ec2:DescribeVolumes"
            ],
            "Resource": "*"
        }
    ]
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Policy to grants service-linked role permissions for Compute Optimization Automation

The following code example shows how to This permission-based policy grants service-linked role permissions for Compute Optimization Automation

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/aco-
automation.amazonaws.com/AWSServiceRoleForComputeOptimizerAutomation",
            "Condition": {"StringLike": {"iam:AWSServiceName": "aco-
automation.amazonaws.com"}}
        },
        {
```

```
            "Effect": "Allow",
            "Action": "iam:PutRolePolicy",
            "Resource": "arn:aws:iam::*:role/aws-service-role/aco-
automation.amazonaws.com/AWSServiceRoleForComputeOptimizerAutomation"
        }
    ]
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Roll back an IAM policy version using an AWS SDK

The following code example shows how to:

- Get the list of policy versions in order by date.
- Find the default policy version.
- Make the previous policy version the default.
- Delete the old default version.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def rollback_policy_version(policy_arn):
    """
    Rolls back to the previous default policy, if it exists.

    1. Gets the list of policy versions in order by date.
    2. Finds the default.
    3. Makes the previous policy the default.
    4. Deletes the old default version.
```

```
    :param policy_arn: The ARN of the policy to roll back.
    :return: The default version of the policy after the rollback.
    """
    try:
        policy_versions = sorted(
            iam.Policy(policy_arn).versions.all(),
            key=operator.attrgetter("create_date"),
        )
        logger.info("Got %s versions for %s.", len(policy_versions), policy_arn)
    except ClientError:
        logger.exception("Couldn't get versions for %s.", policy_arn)
        raise

    default_version = None
    rollback_version = None
    try:
        while default_version is None:
            ver = policy_versions.pop()
            if ver.is_default_version:
                default_version = ver
        rollback_version = policy_versions.pop()
        rollback_version.set_as_default()
        logger.info("Set %s as the default version.",
rollback_version.version_id)
        default_version.delete()
        logger.info("Deleted original default version %s.",
default_version.version_id)
    except IndexError:
        if default_version is None:
            logger.warning("No default version found for %s.", policy_arn)
        elif rollback_version is None:
            logger.warning(
                "Default version %s found for %s, but no previous version exists,
so "
                "nothing to roll back to.",
                default_version.version_id,
                policy_arn,
            )
    except ClientError:
        logger.exception("Couldn't roll back version for %s.", policy_arn)
        raise
    else:
        return rollback_version
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - DeletePolicyVersion
  - ListPolicyVersions
  - SetDefaultPolicyVersion

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Run CPU stress tests on virtual machine instances using fault injection

The following code example shows how to:

- Create IAM roles
- Create a CloudWatch alarm
- Create an experiment template
- Run the experiment
- Verify the results
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS FIS CPU Stress Test Tutorial Script
```

```
# This script automates the steps in the AWS FIS CPU stress test tutorial

#    approach using epoch time calculations that work across all Linux
 distributions

# Set up logging
LOG_FILE="fis-tutorial-$(date +%Y%m%d-%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting AWS FIS CPU Stress Test Tutorial Script"
echo "Logging to $LOG_FILE"
echo "============================================="

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2

    if echo "$output" | grep -i "error" > /dev/null; then
        # Ignore specific expected errors
        if [[ "$cmd" == *"aws fis get-experiment"* ]] && [[ "$output" ==
 *"ConfigurationFailure"* ]]; then
            echo "Note: Experiment failed due to configuration issue. This is
 expected in some cases."
            return 0
        fi

        echo "ERROR: Command failed: $cmd"
        echo "Output: $output"
        cleanup_on_error
        exit 1
    fi
}

# Function to clean up resources on error
cleanup_on_error() {
    echo "Error encountered. Cleaning up resources..."

    if [ -n "$EXPERIMENT_ID" ]; then
        echo "Stopping experiment $EXPERIMENT_ID if running..."
        aws fis stop-experiment --id "$EXPERIMENT_ID" 2>/dev/null || true
    fi

    if [ -n "$TEMPLATE_ID" ]; then
```

```
        echo "Deleting experiment template $TEMPLATE_ID..."
        aws fis delete-experiment-template --id "$TEMPLATE_ID" || true
    fi

    if [ -n "$INSTANCE_ID" ]; then
        echo "Terminating EC2 instance $INSTANCE_ID..."
        aws ec2 terminate-instances --instance-ids "$INSTANCE_ID" || true
    fi

    if [ -n "$ALARM_NAME" ]; then
        echo "Deleting CloudWatch alarm $ALARM_NAME..."
        aws cloudwatch delete-alarms --alarm-names "$ALARM_NAME" || true
    fi

    if [ -n "$INSTANCE_PROFILE_NAME" ]; then
        echo "Removing role from instance profile..."
        aws iam remove-role-from-instance-profile --instance-profile-name
 "$INSTANCE_PROFILE_NAME" --role-name "$EC2_ROLE_NAME" || true

        echo "Deleting instance profile..."
        aws iam delete-instance-profile --instance-profile-name
 "$INSTANCE_PROFILE_NAME" || true
    fi

    if [ -n "$FIS_ROLE_NAME" ]; then
        echo "Deleting FIS role policy..."
        aws iam delete-role-policy --role-name "$FIS_ROLE_NAME" --policy-name
 "$FIS_POLICY_NAME" || true

        echo "Deleting FIS role..."
        aws iam delete-role --role-name "$FIS_ROLE_NAME" || true
    fi

    if [ -n "$EC2_ROLE_NAME" ]; then
        echo "Detaching policy from EC2 role..."
        aws iam detach-role-policy --role-name "$EC2_ROLE_NAME" --policy-arn
 "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore" || true

        echo "Deleting EC2 role..."
        aws iam delete-role --role-name "$EC2_ROLE_NAME" || true
    fi

    echo "Cleanup completed."
}
```

```
# Generate unique identifiers for resources
TIMESTAMP=$(date +%Y%m%d%H%M%S)
FIS_ROLE_NAME="FISRole-${TIMESTAMP}"
FIS_POLICY_NAME="FISPolicy-${TIMESTAMP}"
EC2_ROLE_NAME="EC2SSMRole-${TIMESTAMP}"
INSTANCE_PROFILE_NAME="EC2SSMProfile-${TIMESTAMP}"
ALARM_NAME="FIS-CPU-Alarm-${TIMESTAMP}"

# Track created resources
CREATED_RESOURCES=()

echo "Step 1: Creating IAM role for AWS FIS"
# Create trust policy file for AWS FIS
cat > fis-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "fis.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role for FIS
echo "Creating IAM role $FIS_ROLE_NAME for AWS FIS..."
FIS_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$FIS_ROLE_NAME" \
  --assume-role-policy-document file://fis-trust-policy.json)
check_error "$FIS_ROLE_OUTPUT" "aws iam create-role"
aws iam tag-role --role-name "$FIS_ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-fault-injection-service-gs
CREATED_RESOURCES+=("IAM Role: $FIS_ROLE_NAME")

# Create policy document for SSM actions
cat > fis-ssm-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "ssm:SendCommand",
        "ssm:ListCommands",
        "ssm:ListCommandInvocations"
      ],
      "Resource": "*"
    }
  ]
}
EOF

# Attach policy to the role
echo "Attaching policy $FIS_POLICY_NAME to role $FIS_ROLE_NAME..."
FIS_POLICY_OUTPUT=$(aws iam put-role-policy \
  --role-name "$FIS_ROLE_NAME" \
  --policy-name "$FIS_POLICY_NAME" \
  --policy-document file://fis-ssm-policy.json)
check_error "$FIS_POLICY_OUTPUT" "aws iam put-role-policy"
CREATED_RESOURCES+=("IAM Policy: $FIS_POLICY_NAME attached to $FIS_ROLE_NAME")

echo "Step 2: Creating IAM role for EC2 instance with SSM permissions"
# Create trust policy file for EC2
cat > ec2-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role for EC2
echo "Creating IAM role $EC2_ROLE_NAME for EC2 instance..."
EC2_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$EC2_ROLE_NAME" \
  --assume-role-policy-document file://ec2-trust-policy.json)
```

```
check_error "$EC2_ROLE_OUTPUT" "aws iam create-role"
aws iam tag-role --role-name "$EC2_ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-fault-injection-service-gs
CREATED_RESOURCES+=("IAM Role: $EC2_ROLE_NAME")

# Attach SSM policy to the EC2 role
echo "Attaching AmazonSSMManagedInstanceCore policy to role $EC2_ROLE_NAME..."
EC2_POLICY_OUTPUT=$(aws iam attach-role-policy \
  --role-name "$EC2_ROLE_NAME" \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore)
check_error "$EC2_POLICY_OUTPUT" "aws iam attach-role-policy"
CREATED_RESOURCES+=("IAM Policy: AmazonSSMManagedInstanceCore attached to
 $EC2_ROLE_NAME")

# Create instance profile
echo "Creating instance profile $INSTANCE_PROFILE_NAME..."
PROFILE_OUTPUT=$(aws iam create-instance-profile \
  --instance-profile-name "$INSTANCE_PROFILE_NAME")
check_error "$PROFILE_OUTPUT" "aws iam create-instance-profile"
CREATED_RESOURCES+=("IAM Instance Profile: $INSTANCE_PROFILE_NAME")

# Add role to instance profile
echo "Adding role $EC2_ROLE_NAME to instance profile $INSTANCE_PROFILE_NAME..."
ADD_ROLE_OUTPUT=$(aws iam add-role-to-instance-profile \
  --instance-profile-name "$INSTANCE_PROFILE_NAME" \
  --role-name "$EC2_ROLE_NAME")
check_error "$ADD_ROLE_OUTPUT" "aws iam add-role-to-instance-profile"

# Wait for role to propagate
echo "Waiting for IAM role to propagate..."
sleep 10

echo "Step 3: Launching EC2 instance"
# Get the latest Amazon Linux 2 AMI ID
echo "Finding latest Amazon Linux 2 AMI..."
AMI_ID=$(aws ec2 describe-images \
  --owners amazon \
  --filters "Name=name,Values=amzn2-ami-hvm-*-x86_64-gp2"
 "Name=state,Values=available" \
  --query "sort_by(Images, &CreationDate)[-1].ImageId" \
  --output text)
check_error "$AMI_ID" "aws ec2 describe-images"
echo "Using AMI: $AMI_ID"
```

```
# Launch EC2 instance
echo "Launching EC2 instance with AMI $AMI_ID..."
INSTANCE_OUTPUT=$(aws ec2 run-instances \
  --image-id "$AMI_ID" \
  --instance-type t2.micro \
  --iam-instance-profile Name="$INSTANCE_PROFILE_NAME" \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=FIS-Test-
Instance},{Key=project,Value=doc-smith},{Key=tutorial,Value=aws-fault-injection-
service-gs}]')
check_error "$INSTANCE_OUTPUT" "aws ec2 run-instances"

# Get instance ID
INSTANCE_ID=$(echo "$INSTANCE_OUTPUT" | grep -i "InstanceId" | head -1 | awk -
F'"' '{print $4}')
if [ -z "$INSTANCE_ID" ]; then
    echo "Failed to get instance ID"
    cleanup_on_error
    exit 1
fi
echo "Launched instance: $INSTANCE_ID"
CREATED_RESOURCES+=("EC2 Instance: $INSTANCE_ID")

# Enable detailed monitoring
echo "Enabling detailed monitoring for instance $INSTANCE_ID..."
MONITOR_OUTPUT=$(aws ec2 monitor-instances --instance-ids "$INSTANCE_ID")
check_error "$MONITOR_OUTPUT" "aws ec2 monitor-instances"

# Wait for instance to be running and status checks to pass
echo "Waiting for instance to be ready..."
aws ec2 wait instance-running --instance-ids "$INSTANCE_ID"
aws ec2 wait instance-status-ok --instance-ids "$INSTANCE_ID"
echo "Instance is ready"

echo "Step 4: Creating CloudWatch alarm for CPU utilization"
# Create CloudWatch alarm
echo "Creating CloudWatch alarm $ALARM_NAME..."
ALARM_OUTPUT=$(aws cloudwatch put-metric-alarm \
  --alarm-name "$ALARM_NAME" \
  --alarm-description "Alarm when CPU exceeds 50%" \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Maximum \
  --period 60 \
  --threshold 50 \
```

```
  --comparison-operator GreaterThanOrEqualToThreshold \
  --dimensions "Name=InstanceId,Value=$INSTANCE_ID" \
  --evaluation-periods 1)
check_error "$ALARM_OUTPUT" "aws cloudwatch put-metric-alarm"
CREATED_RESOURCES+=("CloudWatch Alarm: $ALARM_NAME")

# Get the alarm ARN
echo "Getting CloudWatch alarm ARN..."
ALARM_ARN_OUTPUT=$(aws cloudwatch describe-alarms \
  --alarm-names "$ALARM_NAME")
check_error "$ALARM_ARN_OUTPUT" "aws cloudwatch describe-alarms"
ALARM_ARN=$(echo "$ALARM_ARN_OUTPUT" | grep -i "AlarmArn" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$ALARM_ARN" ]; then
    echo "Failed to get alarm ARN"
    cleanup_on_error
    exit 1
fi
echo "Alarm ARN: $ALARM_ARN"

# Wait for the alarm to initialize and reach OK state
echo "Waiting for CloudWatch alarm to initialize (60 seconds)..."
sleep 60

# Check alarm state
echo "Checking alarm state..."
ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms \
  --alarm-names "$ALARM_NAME")
ALARM_STATE=$(echo "$ALARM_STATE_OUTPUT" | grep -i "StateValue" | head -1 | awk -
F'"' '{print $4}')
echo "Current alarm state: $ALARM_STATE"

# If alarm is not in OK state, wait longer or generate some baseline metrics
if [ "$ALARM_STATE" != "OK" ]; then
    echo "Alarm not in OK state. Waiting for alarm to stabilize (additional 60
 seconds)..."
    sleep 60

    # Check alarm state again
    ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms \
      --alarm-names "$ALARM_NAME")
    ALARM_STATE=$(echo "$ALARM_STATE_OUTPUT" | grep -i "StateValue" | head -1 |
 awk -F'"' '{print $4}')
    echo "Updated alarm state: $ALARM_STATE"
```

```
    if [ "$ALARM_STATE" != "OK" ]; then
        echo "Warning: Alarm still not in OK state. Experiment may fail to
 start."
    fi
fi

echo "Step 5: Creating AWS FIS experiment template"
# Get the IAM role ARN
echo "Getting IAM role ARN for $FIS_ROLE_NAME..."
ROLE_ARN_OUTPUT=$(aws iam get-role \
  --role-name "$FIS_ROLE_NAME")
check_error "$ROLE_ARN_OUTPUT" "aws iam get-role"
ROLE_ARN=$(echo "$ROLE_ARN_OUTPUT" | grep -i "Arn" | head -1 | awk -F'"' '{print
 $4}')
if [ -z "$ROLE_ARN" ]; then
    echo "Failed to get role ARN"
    cleanup_on_error
    exit 1
fi
echo "Role ARN: $ROLE_ARN"

# Get account ID and region
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
REGION=$(aws configure get region)
if [ -z "$REGION" ]; then
    REGION="us-east-1"  # Default to us-east-1 if region not set
fi
INSTANCE_ARN="arn:aws:ec2:${REGION}:${ACCOUNT_ID}:instance/${INSTANCE_ID}"
echo "Instance ARN: $INSTANCE_ARN"

# Create experiment template - Fixed JSON escaping issue
cat > experiment-template.json << EOF
{
  "description": "Test CPU stress predefined SSM document",
  "targets": {
    "testInstance": {
      "resourceType": "aws:ec2:instance",
      "resourceArns": ["$INSTANCE_ARN"],
      "selectionMode": "ALL"
    }
  },
  "actions": {
    "runCpuStress": {
```

```
      "actionId": "aws:ssm:send-command",
      "parameters": {
        "documentArn": "arn:aws:ssm:$REGION::document/AWSFIS-Run-CPU-Stress",
        "documentParameters": "{\"DurationSeconds\":\"120\"}",
        "duration": "PT5M"
      },
      "targets": {
        "Instances": "testInstance"
      }
    }
  },
  "stopConditions": [
    {
      "source": "aws:cloudwatch:alarm",
      "value": "$ALARM_ARN"
    }
  ],
  "roleArn": "$ROLE_ARN",
  "tags": {
    "Name": "FIS-CPU-Stress-Experiment",
    "project": "doc-smith",
    "tutorial": "aws-fault-injection-service-gs"
  }
}
EOF

# Create experiment template
echo "Creating AWS FIS experiment template..."
TEMPLATE_OUTPUT=$(aws fis create-experiment-template --cli-input-json file://
experiment-template.json)
check_error "$TEMPLATE_OUTPUT" "aws fis create-experiment-template"
TEMPLATE_ID=$(echo "$TEMPLATE_OUTPUT" | grep -i "id" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$TEMPLATE_ID" ]; then
    echo "Failed to get template ID"
    cleanup_on_error
    exit 1
fi
echo "Experiment template created with ID: $TEMPLATE_ID"
CREATED_RESOURCES+=("FIS Experiment Template: $TEMPLATE_ID")

echo "Step 6: Starting the experiment"
# Start the experiment
echo "Starting AWS FIS experiment using template $TEMPLATE_ID..."
```

```
EXPERIMENT_OUTPUT=$(aws fis start-experiment \
  --experiment-template-id "$TEMPLATE_ID" \
  --tags '{"Name": "FIS-CPU-Stress-Run"}')
check_error "$EXPERIMENT_OUTPUT" "aws fis start-experiment"
EXPERIMENT_ID=$(echo "$EXPERIMENT_OUTPUT" | grep -i "id" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$EXPERIMENT_ID" ]; then
    echo "Failed to get experiment ID"
    cleanup_on_error
    exit 1
fi
echo "Experiment started with ID: $EXPERIMENT_ID"
CREATED_RESOURCES+=("FIS Experiment: $EXPERIMENT_ID")

echo "Step 7: Tracking experiment progress"
# Track experiment progress
echo "Tracking experiment progress..."
MAX_CHECKS=30
CHECK_COUNT=0
EXPERIMENT_STATE=""

while [ $CHECK_COUNT -lt $MAX_CHECKS ]; do
    EXPERIMENT_INFO=$(aws fis get-experiment --id "$EXPERIMENT_ID")
    # Don't check for errors here, as we expect some experiments to fail

    EXPERIMENT_STATE=$(echo "$EXPERIMENT_INFO" | grep -i "status" | head -1 | awk
 -F'"' '{print $4}')
    echo "Experiment state: $EXPERIMENT_STATE"

    if [ "$EXPERIMENT_STATE" == "completed" ] || [ "$EXPERIMENT_STATE" ==
 "stopped" ] || [ "$EXPERIMENT_STATE" == "failed" ]; then
        # Show the reason for the state
        REASON=$(echo "$EXPERIMENT_INFO" | grep -i "reason" | head -1 | awk -F'"'
 '{print $4}')
        if [ -n "$REASON" ]; then
            echo "Reason: $REASON"
        fi
        break
    fi

    echo "Waiting 10 seconds before checking again..."
    sleep 10
    CHECK_COUNT=$((CHECK_COUNT + 1))
done
```

```
if [ $CHECK_COUNT -eq $MAX_CHECKS ]; then
    echo "Experiment is taking longer than expected. You can check its status
 later using:"
    echo "aws fis get-experiment --id $EXPERIMENT_ID"
fi

echo "Step 8: Verifying experiment results"
# Check CloudWatch alarm state
echo "Checking CloudWatch alarm state..."
ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms --alarm-names "$ALARM_NAME")
check_error "$ALARM_STATE_OUTPUT" "aws cloudwatch describe-alarms"
echo "$ALARM_STATE_OUTPUT"

# Get CPU utilization metrics
echo "Getting CPU utilization metrics..."
END_TIME=$(date -u +"%Y-%m-%dT%H:%M:%SZ")

# FIXED: Cross-platform compatible way to calculate time 10 minutes ago
# This approach uses epoch seconds and basic arithmetic which works on all Linux
 distributions
CURRENT_EPOCH=$(date +%s)
TEN_MINUTES_AGO_EPOCH=$((CURRENT_EPOCH - 600))
START_TIME=$(date -u -d "@$TEN_MINUTES_AGO_EPOCH" +"%Y-%m-%dT%H:%M:%SZ" 2>/dev/
null || date -u -r "$TEN_MINUTES_AGO_EPOCH" +"%Y-%m-%dT%H:%M:%SZ")

# Create metric query file
cat > metric-query.json << EOF
[
  {
    "Id": "cpu",
    "MetricStat": {
      "Metric": {
        "Namespace": "AWS/EC2",
        "MetricName": "CPUUtilization",
        "Dimensions": [
          {
            "Name": "InstanceId",
            "Value": "$INSTANCE_ID"
          }
        ]
      },
      "Period": 60,
      "Stat": "Maximum"
```

```
    }
  }
]
EOF

METRICS_OUTPUT=$(aws cloudwatch get-metric-data \
  --start-time "$START_TIME" \
  --end-time "$END_TIME" \
  --metric-data-queries file://metric-query.json)
check_error "$METRICS_OUTPUT" "aws cloudwatch get-metric-data"
echo "CPU Utilization Metrics:"
echo "$METRICS_OUTPUT"

# Display summary of created resources
echo ""
echo "=========================================="
echo "RESOURCES CREATED"
echo "=========================================="
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "- $resource"
done
echo "=========================================="

# Prompt for cleanup
echo ""
echo "=========================================="
echo "CLEANUP CONFIRMATION"
echo "=========================================="
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    echo "Starting cleanup process..."

    # Stop experiment if still running
    if [ "$EXPERIMENT_STATE" != "completed" ] && [ "$EXPERIMENT_STATE" !=
 "stopped" ] && [ "$EXPERIMENT_STATE" != "failed" ]; then
        echo "Stopping experiment $EXPERIMENT_ID..."
        STOP_OUTPUT=$(aws fis stop-experiment --id "$EXPERIMENT_ID")
        check_error "$STOP_OUTPUT" "aws fis stop-experiment"
        echo "Waiting for experiment to stop..."
        sleep 10
    fi
```

```
    # Delete experiment template
    echo "Deleting experiment template $TEMPLATE_ID..."
    DELETE_TEMPLATE_OUTPUT=$(aws fis delete-experiment-template --id
"$TEMPLATE_ID")
    check_error "$DELETE_TEMPLATE_OUTPUT" "aws fis delete-experiment-template"

    # Delete CloudWatch alarm
    echo "Deleting CloudWatch alarm $ALARM_NAME..."
    DELETE_ALARM_OUTPUT=$(aws cloudwatch delete-alarms --alarm-names
"$ALARM_NAME")
    check_error "$DELETE_ALARM_OUTPUT" "aws cloudwatch delete-alarms"

    # Terminate EC2 instance
    echo "Terminating EC2 instance $INSTANCE_ID..."
    TERMINATE_OUTPUT=$(aws ec2 terminate-instances --instance-ids "$INSTANCE_ID")
    check_error "$TERMINATE_OUTPUT" "aws ec2 terminate-instances"
    echo "Waiting for instance to terminate..."
    aws ec2 wait instance-terminated --instance-ids "$INSTANCE_ID"

    # Clean up IAM resources
    echo "Removing role from instance profile..."
    REMOVE_ROLE_OUTPUT=$(aws iam remove-role-from-instance-profile \
      --instance-profile-name "$INSTANCE_PROFILE_NAME" \
      --role-name "$EC2_ROLE_NAME")
    check_error "$REMOVE_ROLE_OUTPUT" "aws iam remove-role-from-instance-profile"

    echo "Deleting instance profile..."
    DELETE_PROFILE_OUTPUT=$(aws iam delete-instance-profile \
      --instance-profile-name "$INSTANCE_PROFILE_NAME")
    check_error "$DELETE_PROFILE_OUTPUT" "aws iam delete-instance-profile"

    echo "Deleting FIS role policy..."
    DELETE_POLICY_OUTPUT=$(aws iam delete-role-policy \
      --role-name "$FIS_ROLE_NAME" \
      --policy-name "$FIS_POLICY_NAME")
    check_error "$DELETE_POLICY_OUTPUT" "aws iam delete-role-policy"

    echo "Detaching policy from EC2 role..."
    DETACH_POLICY_OUTPUT=$(aws iam detach-role-policy \
      --role-name "$EC2_ROLE_NAME" \
      --policy-arn "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore")
    check_error "$DETACH_POLICY_OUTPUT" "aws iam detach-role-policy"

    echo "Deleting FIS role..."
```

```
    DELETE_FIS_ROLE_OUTPUT=$(aws iam delete-role \
      --role-name "$FIS_ROLE_NAME")
    check_error "$DELETE_FIS_ROLE_OUTPUT" "aws iam delete-role"

    echo "Deleting EC2 role..."
    DELETE_EC2_ROLE_OUTPUT=$(aws iam delete-role \
      --role-name "$EC2_ROLE_NAME")
    check_error "$DELETE_EC2_ROLE_OUTPUT" "aws iam delete-role"

    # Clean up temporary files
    echo "Cleaning up temporary files..."
    rm -f fis-trust-policy.json ec2-trust-policy.json fis-ssm-policy.json
 experiment-template.json metric-query.json

    echo "Cleanup completed successfully."
else
    echo "Cleanup skipped. Resources will remain in your AWS account."
    echo "You can manually clean up the resources listed above."
fi

echo ""
echo "Script execution completed."
echo "Log file: $LOG_FILE"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddRoleToInstanceProfile
  - AttachRolePolicy
  - CreateExperimentTemplate
  - CreateInstanceProfile
  - CreateRole
  - DeleteAlarms
  - DeleteExperimentTemplate
  - DeleteInstanceProfile
  - DeleteRole
  - DeleteRolePolicy
  - DescribeAlarms
  - DescribeImages

- DetachRolePolicy
- GetCallerIdentity
- GetExperiment
- GetMetricData
- GetRole
- MonitorInstances
- PutMetricAlarm
- PutRolePolicy
- RemoveRoleFromInstanceProfile
- RunInstances
- StartExperiment
- StopExperiment
- TerminateInstances
- Wait

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Set up Attribute-Based Access Control for DynamoDB using AWS Command Line Interface v2

The following code example shows how to implement Attribute-Based Access Control (ABAC) for DynamoDB.

- Create an IAM policy for ABAC.
- Create tables with tags for different departments.
- List and filter tables based on tags.

Bash

AWS CLI with Bash script

Create an IAM policy for ABAC.

```
# Step 1: Create a policy document for ABAC
cat > abac-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}",
          "aws:ResourceTag/Environment": "Development"
        }
      }
    }
  ]
}
EOF

# Step 2: Create the IAM policy
aws iam create-policy \
    --policy-name DynamoDBDepartmentBasedAccess \
```

```
    --policy-document file://abac-policy.json
```

Create tables with tags for different departments.

```
# Create a DynamoDB table with tags for ABAC
aws dynamodb create-table \
    --table-name FinanceData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Finance \
        Key=Environment,Value=Development

# Create another table with different tags
aws dynamodb create-table \
    --table-name MarketingData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Marketing \
        Key=Environment,Value=Production
```

List and filter tables based on tags.

```
# List all DynamoDB tables
echo "Listing all tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nGetting ARNs for all tables:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# For each table ARN, list its tags
```

```
echo -e "\nListing tags for each table:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    echo -e "\nTags for table: $TABLE_NAME"
    aws dynamodb list-tags-of-resource --resource-arn $ARN
done

# Example: Find tables with a specific tag
echo -e "\nFinding tables with Environment=Production tag:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - CreatePolicy
  - CreateTable
  - ListTables

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Setting up systems management

The following code example shows how to:

- Create IAM permissions for Systems Manager
- Create an IAM role for Systems Manager
- Configure Systems Manager
- Verify the setup
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS Systems Manager Setup Script
# This script sets up AWS Systems Manager for a single account and region
#
# Version 17 fixes:
# 1. Added cloudformation.amazonaws.com to the IAM role trust policy
# 2. Systems Manager Quick Setup uses CloudFormation for deployments, so the role
 must trust CloudFormation service

# Initialize log file
LOG_FILE="ssm_setup_$(date +%Y%m%d_%H%M%S).log"
UNIQUE_ID=$(openssl rand -hex 4)
echo "Starting AWS Systems Manager setup at $(date)" > "$LOG_FILE"

# Function to log commands and their outputs with immediate terminal display
log_cmd() {
    echo "$(date): Running command: $1" | tee -a "$LOG_FILE"
    local output
    output=$(eval "$1" 2>&1)
    local status=$?
    echo "$output" | tee -a "$LOG_FILE"
    return $status
}

# Function to check for errors in command output
check_error() {
    local cmd_output="$1"
    local cmd_status="$2"
    local error_msg="$3"

    if [[ $cmd_status -ne 0 || "$cmd_output" =~ [Ee][Rr][Rr][Oo][Rr] ]]; then
```

```
        echo "ERROR: $error_msg" | tee -a "$LOG_FILE"
        echo "Command output: $cmd_output" | tee -a "$LOG_FILE"
        cleanup_on_error
        exit 1
    fi
}

# Array to track created resources for cleanup
declare -a CREATED_RESOURCES

# Function to add a resource to the tracking array
track_resource() {
    local resource_type="$1"
    local resource_id="$2"
    CREATED_RESOURCES+=("$resource_type:$resource_id")
    echo "Tracked resource: $resource_type:$resource_id" | tee -a "$LOG_FILE"
}

# Function to clean up resources on error
cleanup_on_error() {
    echo "" | tee -a "$LOG_FILE"
    echo "==========================================" | tee -a "$LOG_FILE"
    echo "ERROR OCCURRED - CLEANING UP RESOURCES" | tee -a "$LOG_FILE"
    echo "==========================================" | tee -a "$LOG_FILE"
    echo "The following resources were created:" | tee -a "$LOG_FILE"

    # Display resources in reverse order
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        echo "${CREATED_RESOURCES[$i]}" | tee -a "$LOG_FILE"
    done

    echo "" | tee -a "$LOG_FILE"
    echo "Attempting to clean up resources..." | tee -a "$LOG_FILE"

    # Clean up resources in reverse order
    cleanup_resources
}

# Function to clean up all created resources
cleanup_resources() {
    # Process resources in reverse order (last created, first deleted)
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        IFS=':' read -r resource_type resource_id <<< "${CREATED_RESOURCES[$i]}"
```

```
        echo "Deleting $resource_type: $resource_id" | tee -a "$LOG_FILE"

        case "$resource_type" in
            "IAM_POLICY")
                # Delete the policy (detachment should have been handled when the
 role was deleted)
                log_cmd "aws iam delete-policy --policy-arn $resource_id" || true
                ;;
            "IAM_ROLE")
                # Detach all policies from the role first
                if [[ -n "$POLICY_ARN" ]]; then
                    log_cmd "aws iam detach-role-policy --role-name $resource_id
 --policy-arn $POLICY_ARN" || true
                fi

                # Delete the role
                log_cmd "aws iam delete-role --role-name $resource_id" || true
                ;;
            "SSM_CONFIG_MANAGER")
                log_cmd "aws ssm-quicksetup delete-configuration-manager --
manager-arn $resource_id" || true
                ;;
            *)
                echo "Unknown resource type: $resource_type, cannot delete
 automatically" | tee -a "$LOG_FILE"
                ;;
        esac
    done

    echo "Cleanup completed" | tee -a "$LOG_FILE"

    # Clean up temporary files
    rm -f ssm-onboarding-policy.json trust-policy.json ssm-config.json 2>/dev/
null || true
}

# Main script execution
echo "AWS Systems Manager Setup Script"
echo "================================"
echo "This script will set up AWS Systems Manager for a single account and
 region."
echo "It will create IAM policies and roles, then enable Systems Manager
 features."
echo ""
```

```
# Get the current AWS region
CURRENT_REGION=$(aws configure get region)
if [[ -z "$CURRENT_REGION" ]]; then
    echo "No AWS region configured. Please specify a region:"
    CURRENT_REGION="${AWS_DEFAULT_REGION:-us-west-2}"
    if [[ -z "$CURRENT_REGION" ]]; then
        echo "ERROR: A region must be specified" | tee -a "$LOG_FILE"
        exit 1
    fi
fi

echo "Using AWS region: $CURRENT_REGION" | tee -a "$LOG_FILE"

# Step 1: Create IAM policy for Systems Manager onboarding
echo "Step 1: Creating IAM policy for Systems Manager onboarding..."

# Create policy document
cat > ssm-onboarding-policy.json << 'EOF'
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Sid": "QuickSetupActions",
       "Effect": "Allow",
       "Action": [
         "ssm-quicksetup:*"
       ],
       "Resource": "*"
     },
     {
       "Sid": "SsmReadOnly",
       "Effect": "Allow",
       "Action": [
         "ssm:DescribeAutomationExecutions",
         "ssm:GetAutomationExecution",
         "ssm:ListAssociations",
         "ssm:DescribeAssociation",
         "ssm:ListDocuments",
         "ssm:ListResourceDataSync",
         "ssm:DescribePatchBaselines",
         "ssm:GetPatchBaseline",
         "ssm:DescribeMaintenanceWindows",
         "ssm:DescribeMaintenanceWindowTasks"
```

```
        ],
        "Resource": "*"
      },
      {
        "Sid": "SsmDocument",
        "Effect": "Allow",
        "Action": [
          "ssm:GetDocument",
          "ssm:DescribeDocument"
        ],
        "Resource": [
          "arn:aws:ssm:*:*:document/AWSQuickSetupType-*",
          "arn:aws:ssm:*:*:document/AWS-EnableExplorer"
        ]
      },
      {
        "Sid": "SsmEnableExplorer",
        "Effect": "Allow",
        "Action": "ssm:StartAutomationExecution",
        "Resource": "arn:aws:ssm:*:*:automation-definition/AWS-EnableExplorer:*"
      },
      {
        "Sid": "SsmExplorerRds",
        "Effect": "Allow",
        "Action": [
          "ssm:GetOpsSummary",
          "ssm:CreateResourceDataSync",
          "ssm:UpdateResourceDataSync"
        ],
        "Resource": "arn:aws:ssm:*:*:resource-data-sync/AWS-QuickSetup-*"
      },
      {
        "Sid": "OrgsReadOnly",
        "Effect": "Allow",
        "Action": [
          "organizations:DescribeAccount",
          "organizations:DescribeOrganization",
          "organizations:ListDelegatedAdministrators",
          "organizations:ListRoots",
          "organizations:ListParents",
          "organizations:ListOrganizationalUnitsForParent",
          "organizations:DescribeOrganizationalUnit",
          "organizations:ListAWSServiceAccessForOrganization"
        ],
```

```
      "Resource": "*"
    },
    {
      "Sid": "OrgsAdministration",
      "Effect": "Allow",
      "Action": [
        "organizations:EnableAWSServiceAccess",
        "organizations:RegisterDelegatedAdministrator",
        "organizations:DeregisterDelegatedAdministrator"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "organizations:ServicePrincipal": [
            "ssm.amazonaws.com",
            "ssm-quicksetup.amazonaws.com",
            "member.org.stacksets.cloudformation.amazonaws.com",
            "resource-explorer-2.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid": "CfnReadOnly",
      "Effect": "Allow",
      "Action": [
        "cloudformation:ListStacks",
        "cloudformation:DescribeStacks",
        "cloudformation:ListStackSets",
        "cloudformation:DescribeOrganizationsAccess"
      ],
      "Resource": "*"
    },
    {
      "Sid": "OrgCfnAccess",
      "Effect": "Allow",
      "Action": [
        "cloudformation:ActivateOrganizationsAccess"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CfnStackActions",
      "Effect": "Allow",
```

```
        "Action": [
          "cloudformation:CreateStack",
          "cloudformation:DeleteStack",
          "cloudformation:DescribeStackResources",
          "cloudformation:DescribeStackEvents",
          "cloudformation:GetTemplate",
          "cloudformation:RollbackStack",
          "cloudformation:TagResource",
          "cloudformation:UntagResource",
          "cloudformation:UpdateStack"
        ],
        "Resource": [
          "arn:aws:cloudformation:*:*:stack/StackSet-AWS-QuickSetup-*",
          "arn:aws:cloudformation:*:*:stack/AWS-QuickSetup-*",
          "arn:aws:cloudformation:*:*:type/resource/*"
        ]
      },
      {
        "Sid": "CfnStackSetActions",
        "Effect": "Allow",
        "Action": [
          "cloudformation:CreateStackInstances",
          "cloudformation:CreateStackSet",
          "cloudformation:DeleteStackInstances",
          "cloudformation:DeleteStackSet",
          "cloudformation:DescribeStackInstance",
          "cloudformation:DetectStackSetDrift",
          "cloudformation:ListStackInstanceResourceDrifts",
          "cloudformation:DescribeStackSet",
          "cloudformation:DescribeStackSetOperation",
          "cloudformation:ListStackInstances",
          "cloudformation:ListStackSetOperations",
          "cloudformation:ListStackSetOperationResults",
          "cloudformation:TagResource",
          "cloudformation:UntagResource",
          "cloudformation:UpdateStackSet"
        ],
        "Resource": [
          "arn:aws:cloudformation:*:*:stackset/AWS-QuickSetup-*",
          "arn:aws:cloudformation:*:*:type/resource/*",
          "arn:aws:cloudformation:*:*:stackset-target/AWS-QuickSetup-*:*"
        ]
      },
      {
```

```
        "Sid": "ValidationReadonlyActions",
        "Effect": "Allow",
        "Action": [
          "iam:ListRoles",
          "iam:GetRole"
        ],
        "Resource": "*"
      },
      {
        "Sid": "IamRolesMgmt",
        "Effect": "Allow",
        "Action": [
          "iam:CreateRole",
          "iam:DeleteRole",
          "iam:GetRole",
          "iam:AttachRolePolicy",
          "iam:DetachRolePolicy",
          "iam:GetRolePolicy",
          "iam:ListRolePolicies"
        ],
        "Resource": [
          "arn:aws:iam::*:role/AWS-QuickSetup-*",
          "arn:aws:iam::*:role/service-role/AWS-QuickSetup-*"
        ]
      },
      {
        "Sid": "IamPassRole",
        "Effect": "Allow",
        "Action": [
          "iam:PassRole"
        ],
        "Resource": [
          "arn:aws:iam::*:role/AWS-QuickSetup-*",
          "arn:aws:iam::*:role/service-role/AWS-QuickSetup-*"
        ],
        "Condition": {
          "StringEquals": {
            "iam:PassedToService": [
              "ssm.amazonaws.com",
              "ssm-quicksetup.amazonaws.com",
              "cloudformation.amazonaws.com"
            ]
          }
        }
```

```
    },
    {
      "Sid": "IamRolesPoliciesMgmt",
      "Effect": "Allow",
      "Action": [
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:role/AWS-QuickSetup-*",
        "arn:aws:iam::*:role/service-role/AWS-QuickSetup-*"
      ],
      "Condition": {
        "ArnEquals": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/
AWSSystemsManagerEnableExplorerExecutionPolicy",
            "arn:aws:iam::aws:policy/AWSQuickSetupSSMDeploymentRolePolicy"
          ]
        }
      }
    },
    {
      "Sid": "CfnStackSetsSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/
stacksets.cloudformation.amazonaws.com/
AWSServiceRoleForCloudFormationStackSetsOrgAdmin",
        "arn:aws:iam::*:role/aws-service-role/ssm.amazonaws.com/
AWSServiceRoleForAmazonSSM",
        "arn:aws:iam::*:role/aws-service-role/
accountdiscovery.ssm.amazonaws.com/AWSServiceRoleForAmazonSSM_AccountDiscovery",
        "arn:aws:iam::*:role/aws-service-role/ssm-quicksetup.amazonaws.com/
AWSServiceRoleForSSMQuickSetup",
        "arn:aws:iam::*:role/aws-service-role/resource-explorer-2.amazonaws.com/
AWSServiceRoleForResourceExplorer"
      ]
    }
  ]
}
```

```
EOF

# Create the IAM policy
POLICY_OUTPUT=$(log_cmd "aws iam create-policy --policy-name SSMOnboardingPolicy-
$UNIQUE_ID --policy-document file://ssm-onboarding-policy.json --output json")
POLICY_STATUS=$?
check_error "$POLICY_OUTPUT" $POLICY_STATUS "Failed to create IAM policy"

# Extract the policy ARN
POLICY_ARN=$(echo "$POLICY_OUTPUT" | grep -o 'arn:aws:iam::[0-9]*:policy/
SSMOnboardingPolicy-[a-f0-9]*')
if [[ -z "$POLICY_ARN" ]]; then
    echo "ERROR: Failed to extract policy ARN" | tee -a "$LOG_FILE"
    exit 1
fi

# Track the created policy
track_resource "IAM_POLICY" "$POLICY_ARN"
aws iam tag-policy --policy-arn "$POLICY_ARN" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-systems-manager-gs

echo "Created policy: $POLICY_ARN" | tee -a "$LOG_FILE"

# Step 2: Create and configure IAM role for Systems Manager
echo ""
echo "Step 2: Creating IAM role for Systems Manager..."

# Get current user name
USER_OUTPUT=$(log_cmd "aws sts get-caller-identity --output json")
USER_STATUS=$?
check_error "$USER_OUTPUT" $USER_STATUS "Failed to get caller identity"

# Extract account ID
ACCOUNT_ID=$(echo "$USER_OUTPUT" | grep -o '"Account": "[0-9]*"' | cut -d'"' -f4)
if [[ -z "$ACCOUNT_ID" ]]; then
    echo "ERROR: Failed to extract account ID" | tee -a "$LOG_FILE"
    exit 1
fi

# Generate a unique role name
ROLE_NAME="SSMTutorialRole-$(openssl rand -hex 4)"

# Create trust policy for the role - FIXED: Added cloudformation.amazonaws.com
cat > trust-policy.json << 'EOF'
```

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com",
                    "ssm-quicksetup.amazonaws.com",
                    "cloudformation.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::ACCOUNT_ID:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF

# Replace ACCOUNT_ID placeholder in trust policy
sed -i "s/ACCOUNT_ID/$ACCOUNT_ID/g" trust-policy.json

# Create the IAM role
ROLE_OUTPUT=$(log_cmd "aws iam create-role --role-name $ROLE_NAME --assume-role-
policy-document file://trust-policy.json --description 'Role for Systems Manager
 tutorial' --output json")
ROLE_STATUS=$?
check_error "$ROLE_OUTPUT" $ROLE_STATUS "Failed to create IAM role"

# Extract the role ARN
ROLE_ARN=$(echo "$ROLE_OUTPUT" | grep -o 'arn:aws:iam::[0-9]*:role/[^"]*')
if [[ -z "$ROLE_ARN" ]]; then
    echo "ERROR: Failed to extract role ARN" | tee -a "$LOG_FILE"
    cleanup_on_error
    exit 1
fi

# Track the created role
```

```
track_resource "IAM_ROLE" "$ROLE_NAME"
aws iam tag-role --role-name "$ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-systems-manager-gs

echo "Created IAM role: $ROLE_NAME" | tee -a "$LOG_FILE"
echo "Role ARN: $ROLE_ARN" | tee -a "$LOG_FILE"

# Set identity variables for cleanup
IDENTITY_TYPE="role"
IDENTITY_NAME="$ROLE_NAME"

# Attach the policy to the role
ATTACH_OUTPUT=$(log_cmd "aws iam attach-role-policy --role-name $ROLE_NAME --
policy-arn $POLICY_ARN")
ATTACH_STATUS=$?
check_error "$ATTACH_OUTPUT" $ATTACH_STATUS "Failed to attach policy to role
 $ROLE_NAME"

echo "Policy attached to role: $ROLE_NAME" | tee -a "$LOG_FILE"

# Step 3: Create Systems Manager configuration using Host Management
echo ""
echo "Step 3: Creating Systems Manager configuration..."

# Generate a random identifier for the configuration name
CONFIG_NAME="SSMSetup-$(openssl rand -hex 4)"

# Create configuration file for Systems Manager setup using Host Management
# Added both required parameters for single account deployment based on
 CloudFormation documentation
cat > ssm-config.json << EOF
[
  {
    "Type": "AWSQuickSetupType-SSMHostMgmt",
    "LocalDeploymentAdministrationRoleArn": "$ROLE_ARN",
    "LocalDeploymentExecutionRoleName": "$ROLE_NAME",
    "Parameters": {
      "TargetAccounts": "$ACCOUNT_ID",
      "TargetRegions": "$CURRENT_REGION"
    }
  }
]
EOF
```

```
echo "Configuration file created:" | tee -a "$LOG_FILE"
cat ssm-config.json | tee -a "$LOG_FILE"

# Create the configuration manager
CONFIG_OUTPUT=$(log_cmd "aws ssm-quicksetup create-configuration-manager --name
 \"$CONFIG_NAME\" --configuration-definitions file://ssm-config.json --region
 $CURRENT_REGION")
CONFIG_STATUS=$?
check_error "$CONFIG_OUTPUT" $CONFIG_STATUS "Failed to create Systems Manager
 configuration"

# Extract the manager ARN
MANAGER_ARN=$(echo "$CONFIG_OUTPUT" | grep -o 'arn:aws:ssm-quicksetup:[^"]*')
if [[ -z "$MANAGER_ARN" ]]; then
    echo "ERROR: Failed to extract manager ARN" | tee -a "$LOG_FILE"
    exit 1
fi

# Track the created configuration manager
track_resource "SSM_CONFIG_MANAGER" "$MANAGER_ARN"

echo "Created Systems Manager configuration: $MANAGER_ARN" | tee -a "$LOG_FILE"

# Step 4: Verify the setup
echo ""
echo "Step 4: Verifying the setup..."

# Wait for the configuration to be fully deployed
echo "Waiting for the configuration to be deployed (this may take a few
 minutes)..."
sleep 30

# Check the configuration manager status
VERIFY_OUTPUT=$(log_cmd "aws ssm-quicksetup get-configuration-manager --manager-
arn $MANAGER_ARN --region $CURRENT_REGION")
VERIFY_STATUS=$?
check_error "$VERIFY_OUTPUT" $VERIFY_STATUS "Failed to verify configuration
 manager"

echo "Systems Manager setup completed successfully!" | tee -a "$LOG_FILE"

# List the created resources
echo ""
echo "============================================="
```

```
echo "CREATED RESOURCES"
echo "========================================="
for resource in "${CREATED_RESOURCES[@]+"${CREATED_RESOURCES[@]}"}"; do
    echo "$resource"
done

# Prompt for cleanup
echo ""
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Do you want to clean up all created resources? (y/n): "
CLEANUP_CHOICE="y"

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    echo "Cleaning up resources..." | tee -a "$LOG_FILE"
    cleanup_resources
    echo "Cleanup completed." | tee -a "$LOG_FILE"
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later." | tee -a "$LOG_FILE"
fi

echo ""
echo "Script execution completed. See $LOG_FILE for details."

# Clean up temporary files
rm -f ssm-onboarding-policy.json trust-policy.json ssm-config.json 2>/dev/null ||
 true
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateConfigurationManager
  - CreatePolicy
  - CreateRole
  - DeleteConfigurationManager
  - DeletePolicy
  - DeleteRole
  - DetachRolePolicy

- GetCallerIdentity
- GetConfigurationManager

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Using property variables in monitoring dashboards to monitor multiple serverless functions

The following code example shows how to:

- Create Lambda functions for monitoring
- Create a CloudWatch dashboard
- Add a property variable to the dashboard
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# CloudWatch Dashboard with Lambda Function Variable Script
# This script creates a CloudWatch dashboard with a property variable for Lambda
 function names

set -euo pipefail

# Security: Set restrictive umask
umask 0077
```

```
# Set up logging with secure permissions
LOG_FILE="cloudwatch-dashboard-script-v4.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
echo "Starting script execution at $(date)" >> "$LOG_FILE"

# Function to log commands and their output (with sensitive data sanitization)
log_cmd() {
    local cmd="$1"
    local sanitized_cmd="${cmd//--password*/--password [REDACTED]}"
    sanitized_cmd="${sanitized_cmd//--secret*/--secret [REDACTED]}"
    echo "$(date): Running command: $sanitized_cmd" >> "$LOG_FILE"
    eval "$cmd" 2>&1 | tee -a "$LOG_FILE"
    return ${PIPESTATUS[0]}
}

# Function to check for errors in command output
check_error() {
    local cmd_output="$1"
    local cmd_status="$2"
    local error_msg="$3"

    if [ $cmd_status -ne 0 ] || echo "$cmd_output" | grep -qi "error"; then
        echo "ERROR: $error_msg" | tee -a "$LOG_FILE"
        # Sanitize output before logging
        local sanitized_output="${cmd_output//arn:aws:iam::[0-9]*/
arn:aws:iam::ACCOUNT_ID}"
        echo "Command output: $sanitized_output" | tee -a "$LOG_FILE"
        cleanup_resources
        exit 1
    fi
}

# Trap errors and cleanup
trap 'cleanup_resources' EXIT ERR INT TERM

# Function to clean up resources
cleanup_resources() {
    local exit_code=$?

    echo "" | tee -a "$LOG_FILE"
    echo "========================================" | tee -a "$LOG_FILE"
    echo "CLEANUP PROCESS" | tee -a "$LOG_FILE"
    echo "========================================" | tee -a "$LOG_FILE"
```

```
    if [ -n "${DASHBOARD_NAME:-}" ]; then
        echo "Deleting CloudWatch dashboard: $DASHBOARD_NAME" | tee -a
 "$LOG_FILE"
        aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" 2>&1
 >> "$LOG_FILE" || true
    fi

    if [ -n "${LAMBDA_FUNCTION1:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION1" 2>&1 >>
 "$LOG_FILE" || true
    fi

    if [ -n "${LAMBDA_FUNCTION2:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION2" 2>&1 >>
 "$LOG_FILE" || true
    fi

    if [ -n "${ROLE_NAME:-}" ]; then
        echo "Detaching policy from role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole 2>&1 >>
 "$LOG_FILE" || true

        echo "Deleting IAM role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam delete-role --role-name "$ROLE_NAME" 2>&1 >> "$LOG_FILE" || true
    fi

    # Clean up temporary files securely
    shred -vfz -n 3 trust-policy.json lambda_function.py lambda_function.zip 2>/
dev/null || rm -f trust-policy.json lambda_function.py lambda_function.zip

    echo "Cleanup completed." | tee -a "$LOG_FILE"

    return $exit_code
}

# Validate AWS CLI is installed and authenticated
if ! command -v aws &> /dev/null; then
    echo "ERROR: AWS CLI is not installed" | tee -a "$LOG_FILE"
    exit 1
fi
```

```
if ! aws sts get-caller-identity &> /dev/null; then
    echo "ERROR: AWS CLI is not properly authenticated" | tee -a "$LOG_FILE"
    exit 1
fi

# Get AWS region with validation
AWS_REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$AWS_REGION" ]; then
    AWS_REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $AWS_REGION" | tee -a
 "$LOG_FILE"
else
    echo "Using AWS region: $AWS_REGION" | tee -a "$LOG_FILE"
fi

# Validate region format
if ! [[ "$AWS_REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]$ ]]; then
    echo "ERROR: Invalid AWS region format: $AWS_REGION" | tee -a "$LOG_FILE"
    exit 1
fi

# Generate unique identifiers using secure random with validation
RANDOM_ID=$(openssl rand -hex 6)
if [ -z "$RANDOM_ID" ] || [ ${#RANDOM_ID} -ne 12 ]; then
    echo "ERROR: Failed to generate valid random identifier" | tee -a "$LOG_FILE"
    exit 1
fi

DASHBOARD_NAME="LambdaMetricsDashboard-${RANDOM_ID}"
LAMBDA_FUNCTION1="TestFunction1-${RANDOM_ID}"
LAMBDA_FUNCTION2="TestFunction2-${RANDOM_ID}"
ROLE_NAME="LambdaExecutionRole-${RANDOM_ID}"

# Validate resource names don't exceed AWS limits
if [ ${#DASHBOARD_NAME} -gt 128 ] || [ ${#LAMBDA_FUNCTION1} -gt 64 ] ||
 [ ${#ROLE_NAME} -gt 64 ]; then
    echo "ERROR: Generated resource names exceed AWS limits" | tee -a "$LOG_FILE"
    exit 1
fi

echo "Using random identifier: $RANDOM_ID" | tee -a "$LOG_FILE"
echo "Dashboard name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
```

```
echo "Lambda function names: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
 "$LOG_FILE"
echo "IAM role name: $ROLE_NAME" | tee -a "$LOG_FILE"

# Create IAM role for Lambda functions
echo "Creating IAM role for Lambda..." | tee -a "$LOG_FILE"
TRUST_POLICY='{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}'

echo "$TRUST_POLICY" > trust-policy.json
chmod 600 trust-policy.json

# Validate JSON before use
if ! python3 -m json.tool trust-policy.json > /dev/null 2>&1; then
    echo "ERROR: Invalid trust policy JSON" | tee -a "$LOG_FILE"
    exit 1
fi

ROLE_OUTPUT=$(log_cmd "aws iam create-role --role-name '$ROLE_NAME' --assume-
role-policy-document file://trust-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=cloudwatch-streams --output json")
check_error "$ROLE_OUTPUT" $? "Failed to create IAM role"

ROLE_ARN=$(echo "$ROLE_OUTPUT" | python3 -c "import sys, json;
 print(json.load(sys.stdin)['Role']['Arn'])" 2>/dev/null)
if [ -z "$ROLE_ARN" ]; then
    echo "ERROR: Failed to extract Role ARN" | tee -a "$LOG_FILE"
    exit 1
fi
echo "Role ARN: $ROLE_ARN" | tee -a "$LOG_FILE"

# Attach Lambda basic execution policy to the role
echo "Attaching Lambda execution policy to role..." | tee -a "$LOG_FILE"
```

```
POLICY_OUTPUT=$(log_cmd "aws iam attach-role-policy --role-name '$ROLE_NAME' --
policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole")
check_error "$POLICY_OUTPUT" $? "Failed to attach policy to role"

# Wait for role to propagate
echo "Waiting for IAM role to propagate..." | tee -a "$LOG_FILE"
sleep 10

# Create simple Python Lambda function code with security validation
echo "Creating Lambda function code..." | tee -a "$LOG_FILE"
cat > lambda_function.py << 'LAMBDA_EOF'
def handler(event, context):
    print("Lambda function executed successfully")
    return {
        'statusCode': 200,
        'body': 'Success'
    }
LAMBDA_EOF

chmod 600 lambda_function.py

# Validate Python syntax
if ! python3 -m py_compile lambda_function.py 2>/dev/null; then
    echo "ERROR: Invalid Python syntax in Lambda function" | tee -a "$LOG_FILE"
    exit 1
fi

# Zip the Lambda function code
zip -j -q lambda_function.zip lambda_function.py
if [ ! -f lambda_function.zip ]; then
    echo "ERROR: Failed to create lambda_function.zip" | tee -a "$LOG_FILE"
    exit 1
fi
chmod 600 lambda_function.zip

# Validate zip file integrity
if ! unzip -t lambda_function.zip > /dev/null 2>&1; then
    echo "ERROR: Created zip file is corrupted" | tee -a "$LOG_FILE"
    exit 1
fi

# Create first Lambda function
echo "Creating first Lambda function: $LAMBDA_FUNCTION1..." | tee -a "$LOG_FILE"
```

```
LAMBDA1_OUTPUT=$(log_cmd "aws lambda create-function --function-name
 '$LAMBDA_FUNCTION1' --runtime python3.11 --role '$ROLE_ARN' --handler
 lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA1_OUTPUT" $? "Failed to create first Lambda function"

# Create second Lambda function
echo "Creating second Lambda function: $LAMBDA_FUNCTION2..." | tee -a "$LOG_FILE"
LAMBDA2_OUTPUT=$(log_cmd "aws lambda create-function --function-name
 '$LAMBDA_FUNCTION2' --runtime python3.11 --role '$ROLE_ARN' --handler
 lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA2_OUTPUT" $? "Failed to create second Lambda function"

# Invoke Lambda functions to generate some metrics
echo "Invoking Lambda functions to generate metrics..." | tee -a "$LOG_FILE"
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION1' --payload '{}' /
dev/null" || true
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION2' --payload '{}' /
dev/null" || true

# Create CloudWatch dashboard with property variable
echo "Creating CloudWatch dashboard with property variable..." | tee -a
 "$LOG_FILE"

# Create dashboard body with proper escaping and validation
DASHBOARD_BODY=$(cat <<EOF
{
  "widgets": [
    {
      "type": "metric",
      "x": 0,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "$LAMBDA_FUNCTION1" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$AWS_REGION",
        "title": "Lambda Invocations",
        "period": 300,
```

```
        "stat": "Sum"
      }
    }
  ]
}
EOF
)

# Validate JSON before sending
if ! echo "$DASHBOARD_BODY" | python3 -m json.tool > /dev/null 2>&1; then
    echo "ERROR: Dashboard body is not valid JSON" | tee -a "$LOG_FILE"
    exit 1
fi

# First create a basic dashboard without variables
echo "Creating initial dashboard without variables..." | tee -a "$LOG_FILE"
DASHBOARD_OUTPUT=$(aws cloudwatch put-dashboard --dashboard-name
 "$DASHBOARD_NAME" --dashboard-body "$DASHBOARD_BODY" --output json 2>&1)
check_error "$DASHBOARD_OUTPUT" $? "Failed to create initial CloudWatch
 dashboard"

# Now let's try to add a property variable using the console instructions
echo "To complete the tutorial, please follow these steps in the CloudWatch
 console:" | tee -a "$LOG_FILE"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/" | tee -a "$LOG_FILE"
echo "2. Navigate to Dashboards and select your dashboard: $DASHBOARD_NAME" | tee
 -a "$LOG_FILE"
echo "3. Choose Actions > Variables > Create a variable" | tee -a "$LOG_FILE"
echo "4. Choose Property variable" | tee -a "$LOG_FILE"
echo "5. For Property that the variable changes, choose FunctionName" | tee -a
 "$LOG_FILE"
echo "6. For Input type, choose Select menu (dropdown)" | tee -a "$LOG_FILE"
echo "7. Choose Use the results of a metric search" | tee -a "$LOG_FILE"
echo "8. Choose Pre-built queries > Lambda > Errors" | tee -a "$LOG_FILE"
echo "9. Choose By Function Name and then choose Search" | tee -a "$LOG_FILE"
echo "10. (Optional) Configure any secondary settings as desired" | tee -a
 "$LOG_FILE"
echo "11. Choose Add variable" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "The dashboard has been created and can be accessed at:" | tee -a
 "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"
```

```
# Verify dashboard creation
echo "Verifying dashboard creation..." | tee -a "$LOG_FILE"
VERIFY_OUTPUT=$(aws cloudwatch get-dashboard --dashboard-name "$DASHBOARD_NAME"
 --output json 2>&1)
check_error "$VERIFY_OUTPUT" $? "Failed to verify dashboard creation"

echo "" | tee -a "$LOG_FILE"
echo "==========================================" | tee -a "$LOG_FILE"
echo "DASHBOARD CREATED SUCCESSFULLY" | tee -a "$LOG_FILE"
echo "==========================================" | tee -a "$LOG_FILE"
echo "Dashboard Name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "Lambda Functions: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
 "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "You can view your dashboard in the CloudWatch console:" | tee -a
 "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"

# Auto-confirm cleanup
echo "" | tee -a "$LOG_FILE"
echo "==========================================" | tee -a "$LOG_FILE"
echo "CLEANUP CONFIRMATION" | tee -a "$LOG_FILE"
echo "==========================================" | tee -a "$LOG_FILE"
echo "The following resources were created:" | tee -a "$LOG_FILE"
echo "- CloudWatch Dashboard: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
echo "- IAM Role: $ROLE_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "Auto-confirming cleanup of all created resources..." | tee -a "$LOG_FILE"

echo "Script completed successfully." | tee -a "$LOG_FILE"
exit 0
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateFunction
  - CreateRole
  - DeleteDashboards

- DeleteFunction
- DeleteRole
- DetachRolePolicy
- GetDashboard
- Invoke
- PutDashboard

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Work with DynamoDB Streams and Time-to-Live using AWS Command Line Interface v2

The following code example shows how to manage DynamoDB Streams and Time-to-Live features.

- Create a table with Streams enabled.
- Describe Streams.
- Create a Lambda function for processing Streams.
- Enable TTL on a table.
- Add items with TTL attributes.
- Describe TTL settings.

Bash

AWS CLI with Bash script

Create a table with Streams enabled.

```
# Create a table with DynamoDB Streams enabled
aws dynamodb create-table \
    --table-name StreamsDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
```

```
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

Describe Streams.

```
# Get information about the stream
aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.StreamSpecification"

# Get the stream ARN
STREAM_ARN=$(aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.LatestStreamArn" \
    --output text)

echo "Stream ARN: $STREAM_ARN"

# Describe the stream
aws dynamodbstreams describe-stream \
    --stream-arn $STREAM_ARN
```

Create a Lambda function for Streams.

```
# Step 1: Create an IAM role for the Lambda function
cat > trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

aws iam create-role \
    --role-name DynamoDBStreamsLambdaRole \
```

```
    --assume-role-policy-document file://trust-policy.json

# Step 2: Attach permissions to the role
aws iam attach-role-policy \
    --role-name DynamoDBStreamsLambdaRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaDynamoDBExecutionRole

# Step 3: Create a Lambda function (code would be in a separate file)
echo "Lambda function creation would be done separately with appropriate code"

# Step 4: Create an event source mapping
echo "Example command to create event source mapping:"
echo "aws lambda create-event-source-mapping \\"
echo "    --function-name ProcessDynamoDBRecords \\"
echo "    --event-source $STREAM_ARN \\"
echo "    --batch-size 100 \\"
echo "    --starting-position LATEST"
```

Enable TTL on a table.

```
# Create a table for TTL demonstration
aws dynamodb create-table \
    --table-name TTLDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST

# Wait for table to become active
aws dynamodb wait table-exists --table-name TTLDemo

# Enable TTL on the table
aws dynamodb update-time-to-live \
    --table-name TTLDemo \
    --time-to-live-specification "Enabled=true, AttributeName=ExpirationTime"
```

Add items with TTL attributes.

```
# Calculate expiration time (current time + 1 day in seconds)
```

```
EXPIRATION_TIME=$(date -d "+1 day" +%s)

# Add an item with TTL attribute
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item1"},
        "Data": {"S": "This item will expire in 1 day"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME'"}
    }'

# Add an item that expires in 1 hour
EXPIRATION_TIME_HOUR=$(date -d "+1 hour" +%s)
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item2"},
        "Data": {"S": "This item will expire in 1 hour"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME_HOUR'"}
    }'
```

Describe TTL settings.

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Work with the IAM Policy Builder API using an AWS SDK

The following code example shows how to:

- Create IAM policies by using the object-oriented API.
- Use the IAM Policy Builder API with the IAM service.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The examples use the following imports.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.policybuilder.iam.IamConditionOperator;
import software.amazon.awssdk.policybuilder.iam.IamEffect;
import software.amazon.awssdk.policybuilder.iam.IamPolicy;
import software.amazon.awssdk.policybuilder.iam.IamPolicyWriter;
import software.amazon.awssdk.policybuilder.iam.IamPrincipal;
import software.amazon.awssdk.policybuilder.iam.IamPrincipalType;
import software.amazon.awssdk.policybuilder.iam.IamResource;
import software.amazon.awssdk.policybuilder.iam.IamStatement;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iam.IamClient;
import software.amazon.awssdk.services.iam.model.GetPolicyResponse;
import software.amazon.awssdk.services.iam.model.GetPolicyVersionResponse;
import software.amazon.awssdk.services.sts.StsClient;

import java.net.URLDecoder;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.List;
```

Create a time-based policy.

```
        public String timeBasedPolicyExample() {
                IamPolicy policy = IamPolicy.builder()
                                .addStatement(b -> b
                                                .effect(IamEffect.ALLOW)
                                                .addAction("dynamodb:GetItem")
                                                .addResource(IamResource.ALL)
                                                .addCondition(b1 -> b1

 .operator(IamConditionOperator.DATE_GREATER_THAN)

 .key("aws:CurrentTime")

 .value("2020-04-01T00:00:00Z"))
                                                .addCondition(b1 -> b1

 .operator(IamConditionOperator.DATE_LESS_THAN)

 .key("aws:CurrentTime")

 .value("2020-06-30T23:59:59Z")))
                                .build();

                // Use an IamPolicyWriter to write out the JSON string to a more
 readable
                // format.
                return policy.toJson(IamPolicyWriter.builder()
                                .prettyPrint(true)
                                .build());
        }
```

Create a policy with multiple conditions.

```
        public String multipleConditionsExample() {
                IamPolicy policy = IamPolicy.builder()
                                .addStatement(b -> b
```

```
                                                .effect(IamEffect.ALLOW)
                                                .addAction("dynamodb:GetItem")

 .addAction("dynamodb:BatchGetItem")
                                                .addAction("dynamodb:Query")
                                                .addAction("dynamodb:PutItem")
                                                .addAction("dynamodb:UpdateItem")
                                                .addAction("dynamodb:DeleteItem")

 .addAction("dynamodb:BatchWriteItem")

 .addResource("arn:aws:dynamodb:*:*:table/table-name")

 .addConditions(IamConditionOperator.STRING_EQUALS

 .addPrefix("ForAllValues:"),

 "dynamodb:Attributes",
                                                                    List.of("column-
name1", "column-name2", "column-name3"))
                                                .addCondition(b1 -> b1

 .operator(IamConditionOperator.STRING_EQUALS

 .addSuffix("IfExists"))

 .key("dynamodb:Select")

 .value("SPECIFIC_ATTRIBUTES")))
                                .build();

                return policy.toJson(IamPolicyWriter.builder()
                                .prettyPrint(true).build());
        }
```

Use principals in a policy.

```
        public String specifyPrincipalsExample() {
                IamPolicy policy = IamPolicy.builder()
                                .addStatement(b -> b
                                                .effect(IamEffect.DENY)
                                                .addAction("s3:*")
```

```
                                                .addPrincipal(IamPrincipal.ALL)
                                                .addResource("arn:aws:s3:::amzn-
s3-demo-bucket/*")
                                                .addResource("arn:aws:s3:::amzn-
s3-demo-bucket")
                                                .addCondition(b1 -> b1

 .operator(IamConditionOperator.ARN_NOT_EQUALS)

 .key("aws:PrincipalArn")

 .value("arn:aws:iam::444455556666:user/user-name")))
                                .build();
                return policy.toJson(IamPolicyWriter.builder()
                                .prettyPrint(true).build());
        }
```

Allow cross-account access.

```
        public String allowCrossAccountAccessExample() {
                IamPolicy policy = IamPolicy.builder()
                                .addStatement(b -> b
                                                .effect(IamEffect.ALLOW)

 .addPrincipal(IamPrincipalType.AWS, "111122223333")
                                                .addAction("s3:PutObject")
                                                .addResource("arn:aws:s3:::amzn-
s3-demo-bucket/*")
                                                .addCondition(b1 -> b1

 .operator(IamConditionOperator.STRING_EQUALS)
                                                                        .key("s3:x-amz-
acl")
                                                                        .value("bucket-
owner-full-control")))
                                .build();
                return policy.toJson(IamPolicyWriter.builder()
                                .prettyPrint(true).build());
        }
```

Build and upload an `IamPolicy`.

```
       public String createAndUploadPolicyExample(IamClient iam, String
 accountID, String policyName) {
               // Build the policy.
               IamPolicy policy = IamPolicy.builder() // 'version' defaults to
 "2012-10-17".
                               .addStatement(IamStatement.builder()
                                               .effect(IamEffect.ALLOW)
                                               .addAction("dynamodb:PutItem")

 .addResource("arn:aws:dynamodb:us-east-1:" + accountID
                                                                       + ":table/
exampleTableName")
                                               .build())
                               .build();
               // Upload the policy.
               iam.createPolicy(r ->
 r.policyName(policyName).policyDocument(policy.toJson()));
               return
 policy.toJson(IamPolicyWriter.builder().prettyPrint(true).build());
       }
```

Download and work with an `IamPolicy`.

```
       public String createNewBasedOnExistingPolicyExample(IamClient iam, String
 accountID, String policyName,
                       String newPolicyName) {

               String policyArn = "arn:aws:iam::" + accountID + ":policy/" +
 policyName;
               GetPolicyResponse getPolicyResponse = iam.getPolicy(r ->
 r.policyArn(policyArn));

               String policyVersion =
 getPolicyResponse.policy().defaultVersionId();
               GetPolicyVersionResponse getPolicyVersionResponse = iam
                               .getPolicyVersion(r ->
 r.policyArn(policyArn).versionId(policyVersion));

               // Create an IamPolicy instance from the JSON string returned
 from IAM.
               String decodedPolicy =
 URLDecoder.decode(getPolicyVersionResponse.policyVersion().document(),
```

```
                                StandardCharsets.UTF_8);
                IamPolicy policy = IamPolicy.fromJson(decodedPolicy);

                /*
                 * All IamPolicy components are immutable, so use the copy method
that creates a
                 * new instance that
                 * can be altered in the same method call.
                 *
                 * Add the ability to get an item from DynamoDB as an additional
action.
                 */
                IamStatement newStatement = policy.statements().get(0).copy(s ->
s.addAction("dynamodb:GetItem"));

                // Create a new statement that replaces the original statement.
                IamPolicy newPolicy = policy.copy(p ->
p.statements(Arrays.asList(newStatement)));

                // Upload the new policy. IAM now has both policies.
                iam.createPolicy(r -> r.policyName(newPolicyName)
                                .policyDocument(newPolicy.toJson()));

                return
newPolicy.toJson(IamPolicyWriter.builder().prettyPrint(true).build());
        }
```

- For more information, see AWS SDK for Java 2.x Developer Guide.
- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreatePolicy
  - GetPolicy
  - GetPolicyVersion

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples for AWS STS using AWS SDKs

The following code examples show how to use AWS STS with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

AWS STS

- Basic examples for AWS STS using AWS SDKs
  - Actions for AWS STS using AWS SDKs
    - Use AssumeRole with an AWS SDK or CLI
    - Use AssumeRoleWithWebIdentity with a CLI
    - Use DecodeAuthorizationMessage with a CLI
    - Use GetFederationToken with a CLI
    - Use GetSessionToken with an AWS SDK or CLI
- Scenarios for AWS STS using AWS SDKs
  - Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK
  - Configure container service connectivity
  - Construct a URL with AWS STS for federated users using an AWS SDK
  - Create a container task for the serverless launch type
  - Create a rest API with function proxy integration
  - Creating a container service for virtual machine instances
  - Creating a managed monitoring workspace
  - Creating a monitoring dashboard with function name as a variable
  - Get a session token that requires an MFA token with AWS STS using an AWS SDK
  - Getting started with configuration management

- Getting started with container registries
- Getting started with internet of things messaging
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with push notifications
- Getting started with search and analytics engines
- Getting started with web application firewalls
- Moving hardcoded secrets to secure secret storage
- Run CPU stress tests on virtual machine instances using fault injection
- Setting up systems management

Basic examples for AWS STS using AWS SDKs

The following code examples show how to use the basics of AWS Security Token Service with AWS SDKs.

Examples

- Actions for AWS STS using AWS SDKs
  - Use AssumeRole with an AWS SDK or CLI
  - Use AssumeRoleWithWebIdentity with a CLI
  - Use DecodeAuthorizationMessage with a CLI
  - Use GetFederationToken with a CLI
  - Use GetSessionToken with an AWS SDK or CLI

Actions for AWS STS using AWS SDKs

The following code examples demonstrate how to perform individual AWS STS actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the AWS STS API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for AWS STS using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the AWS Security Token Service API Reference.

Examples

- Use AssumeRole with an AWS SDK or CLI
- Use AssumeRoleWithWebIdentity with a CLI
- Use DecodeAuthorizationMessage with a CLI
- Use GetFederationToken with a CLI
- Use GetSessionToken with an AWS SDK or CLI

Use `AssumeRole` with an AWS SDK or CLI

The following code examples show how to use `AssumeRole`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Assume an IAM role that requires an MFA token
- Construct a URL for federated users

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace AssumeRoleExample
{
```

```
    class AssumeRole
    {
        /// <summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html
        /// for help in working with roles.
        /// </summary>

        // A region property may be used if the profile or credentials loaded do
 not specify a region,
        // or to use a specific region.
        private static readonly RegionEndpoint REGION = RegionEndpoint.USWest2;

        static async Task Main()
        {
            // Create the SecurityToken client and then display the identity of
 the
            // default user.
            var roleArnToAssume = "arn:aws:iam::123456789012:role/
testAssumeRole";

            var client = new
 Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

            // Get and display the information about the identity of the default
 user.
            var callerIdRequest = new GetCallerIdentityRequest();
            var caller = await client.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"Original Caller: {caller.Arn}");

            // Create the request to use with the AssumeRoleAsync call.
            var assumeRoleReq = new AssumeRoleRequest()
            {
                DurationSeconds = 1600,
                RoleSessionName = "Session1",
```

```
                RoleArn = roleArnToAssume
            };

            var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

            // Now create a new client based on the credentials of the caller
 assuming the role.
            var client2 = new AmazonSecurityTokenServiceClient(credentials:
 assumeRoleRes.Credentials, REGION);

            // Get and display information about the caller that has assumed the
 defined role.
            var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"AssumedRole Caller: {caller2.Arn}");
        }
    }
}
```

- For API details, see AssumeRole in *AWS SDK for .NET API Reference*.

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
#############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
#############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}
```

```
##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function sts_assume_role
#
# This function assumes a role in the AWS account and returns the temporary
#  credentials.
#
# Parameters:
#       -n role_session_name -- The name of the session.
#       -r role_arn -- The ARN of the role to assume.
#
# Returns:
#       [access_key_id, secret_access_key, session_token]
#     And:
#       0 - If successful.
#       1 - If an error occurred.
##############################################################################
function sts_assume_role() {
  local role_session_name role_arn response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function sts_assume_role"
    echo "Assumes a role in the AWS account and returns the temporary
 credentials:"
    echo "  -n role_session_name -- The name of the session."
    echo "  -r role_arn -- The ARN of the role to assume."
    echo ""
  }

  while getopts n:r:h option; do
    case "${option}" in
      n) role_session_name=${OPTARG} ;;
      r) role_arn=${OPTARG} ;;
```

```
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done

  response=$(aws sts assume-role \
    --role-session-name "$role_session_name" \
    --role-arn "$role_arn" \
    --output text \
    --query "Credentials.[AccessKeyId, SecretAccessKey, SessionToken]")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-role operation failed.\n$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

- For API details, see AssumeRole in *AWS CLI Command Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
bool AwsDoc::STS::assumeRole(const Aws::String &roleArn,
                             const Aws::String &roleSessionName,
                             const Aws::String &externalId,
                             Aws::Auth::AWSCredentials &credentials,
                             const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::STS::STSClient sts(clientConfig);
    Aws::STS::Model::AssumeRoleRequest sts_req;

    sts_req.SetRoleArn(roleArn);
    sts_req.SetRoleSessionName(roleSessionName);
    sts_req.SetExternalId(externalId);

    const Aws::STS::Model::AssumeRoleOutcome outcome = sts.AssumeRole(sts_req);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error assuming IAM role. " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Credentials successfully retrieved." << std::endl;
        const Aws::STS::Model::AssumeRoleResult result = outcome.GetResult();
        const Aws::STS::Model::Credentials &temp_credentials =
 result.GetCredentials();

        // Store temporary credentials in return argument.
        // Note: The credentials object returned by assumeRole differs
        // from the AWSCredentials object used in most situations.
        credentials.SetAWSAccessKeyId(temp_credentials.GetAccessKeyId());
        credentials.SetAWSSecretKey(temp_credentials.GetSecretAccessKey());
        credentials.SetSessionToken(temp_credentials.GetSessionToken());
    }

    return outcome.IsSuccess();
}
```

- For API details, see AssumeRole in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To assume a role

The following `assume-role` example retrieves a set of short-term credentials for the IAM role `s3-access-example`.

```
aws sts assume-role \
    --role-arn arn:aws:iam::123456789012:role/xaccounts3access \
    --role-session-name s3-access-example
```

Output:

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA3XFRBF535PLBIFPI4:s3-access-example",
        "Arn": "arn:aws:sts::123456789012:assumed-role/xaccounts3access/s3-
access-example"
    },
    "Credentials": {
        "SecretAccessKey": "9drTJvcXLB89EXAMPLELB8923FB892xMFI",
        "SessionToken": "AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/
qwjzP2iEXAMPLEbw/m3hsj8VBTkPORGvr9jM5sgP+w9IZWZnU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYlFVgAUj
+7Indz3LU0aTWk1WKIjHmmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nyzM
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFiPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8B
IcrxSpnWEXAMPLEXSDFTAQAM6Dl9zR0tXoybnlrZIwMLlMi1Kcgo5OytwU=",
        "Expiration": "2016-03-15T00:05:07Z",
        "AccessKeyId": "ASIAJEXAMPLEXEG2JICEA"
    }
}
```

The output of the command contains an access key, secret key, and session token that you can use to authenticate to AWS.

For AWS CLI use, you can set up a named profile associated with a role. When you use the profile, the AWS CLI will call assume-role and manage credentials for you. For more information, see Use an IAM role in the AWS CLI in the *AWS CLI User Guide*.

- For API details, see AssumeRole in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sts.StsClient;
import software.amazon.awssdk.services.sts.model.AssumeRoleRequest;
import software.amazon.awssdk.services.sts.model.StsException;
import software.amazon.awssdk.services.sts.model.AssumeRoleResponse;
import software.amazon.awssdk.services.sts.model.Credentials;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * To make this code example work, create a Role that you want to assume.
 * Then define a Trust Relationship in the AWS Console. You can use this as an
 * example:
 *
 * {
 * "Version":"2012-10-17",
 * "Statement": [
 * {
 * "Effect": "Allow",
 * "Principal": {
 * "AWS": "<Specify the ARN of your IAM user you are using in this code example>"
 * },
 * "Action": "sts:AssumeRole"
 * }
 * ]
 * }
 *
 * For more information, see "Editing the Trust Relationship for an Existing
 * Role" in the AWS Directory Service guide.
```

```
 *
 * Also, set up your development environment, including your credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AssumeRole {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <roleArn> <roleSessionName>\s

                Where:
                    roleArn - The Amazon Resource Name (ARN) of the role to
 assume (for example, arn:aws:iam::000008047983:role/s3role).\s
                    roleSessionName - An identifier for the assumed role session
 (for example, mysession).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String roleArn = args[0];
        String roleSessionName = args[1];
        Region region = Region.US_EAST_1;
        StsClient stsClient = StsClient.builder()
                .region(region)
                .build();

        assumeGivenRole(stsClient, roleArn, roleSessionName);
        stsClient.close();
    }

    public static void assumeGivenRole(StsClient stsClient, String roleArn,
 String roleSessionName) {
        try {
            AssumeRoleRequest roleRequest = AssumeRoleRequest.builder()
                    .roleArn(roleArn)
                    .roleSessionName(roleSessionName)
```

```
                    .build();

            AssumeRoleResponse roleResponse = stsClient.assumeRole(roleRequest);
            Credentials myCreds = roleResponse.credentials();

            // Display the time when the temp creds expire.
            Instant exTime = myCreds.expiration();
            String tokenInfo = myCreds.sessionToken();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
 DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.US)
                    .withZone(ZoneId.systemDefault());

            formatter.format(exTime);
            System.out.println("The token " + tokenInfo + "  expires on " +
 exTime);

        } catch (StsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see AssumeRole in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client.

```
import { STSClient } from "@aws-sdk/client-sts";
```

```
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Assume the IAM role.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- For API details, see AssumeRole in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
  DurationSeconds: 900,
};
var roleCreds;

// Create the STS service object
var sts = new AWS.STS({ apiVersion: "2011-06-15" });

//Assume Role
sts.assumeRole(roleToAssume, function (err, data) {
  if (err) console.log(err, err.stack);
  else {
    roleCreds = {
      accessKeyId: data.Credentials.AccessKeyId,
      secretAccessKey: data.Credentials.SecretAccessKey,
      sessionToken: data.Credentials.SessionToken,
    };
    stsGetCallerIdentity(roleCreds);
  }
});

//Get Arn of current identity
function stsGetCallerIdentity(creds) {
  var stsParams = { credentials: creds };
  // Create STS service object
  var sts = new AWS.STS(stsParams);
```

```
  sts.getCallerIdentity({}, function (err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data.Arn);
    }
  });
}
```

- For API details, see AssumeRole in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns a set of temporary credentials (access key, secret key and session token) that can be used for one hour to access AWS resources that the requesting user might not normally have access to. The returned credentials have the permissions that are allowed by the access policy of the role being assumed and the policy that was supplied (you cannot use the supplied policy to grant permissions in excess of those defined by the access policy of the role being assumed).

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -Policy "...JSON policy..." -DurationInSeconds 3600
```

Example 2: Returns a set of temporary credentials, valid for one hour, that have the same permissions that are defined in the access policy of the role being assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600
```

Example 3: Returns a set of temporary credentials supplying the serial number and generated token from an MFA associated with the user credentials used to execute the cmdlet.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -SerialNumber "GAHT12345678" -TokenCode "123456"
```

Example 4: Returns a set of temporary credentials that have assumed a role defined in a customer account. For each role that the third party can assume, the customer account must create a role using an identifier that must be passed in the -ExternalId parameter each time the role is assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -ExternalId "ABC123"
```

- For API details, see AssumeRole in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns a set of temporary credentials (access key, secret key and session token) that can be used for one hour to access AWS resources that the requesting user might not normally have access to. The returned credentials have the permissions that are allowed by the access policy of the role being assumed and the policy that was supplied (you cannot use the supplied policy to grant permissions in excess of those defined by the access policy of the role being assumed).

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -Policy "...JSON policy..." -DurationInSeconds 3600
```

Example 2: Returns a set of temporary credentials, valid for one hour, that have the same permissions that are defined in the access policy of the role being assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600
```

Example 3: Returns a set of temporary credentials supplying the serial number and generated token from an MFA associated with the user credentials used to execute the cmdlet.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -SerialNumber "GAHT12345678" -TokenCode "123456"
```

Example 4: Returns a set of temporary credentials that have assumed a role defined in a customer account. For each role that the third party can assume, the customer account must create a role using an identifier that must be passed in the -ExternalId parameter each time the role is assumed.

```
Use-STSRole -RoleSessionName "Bob" -RoleArn "arn:aws:iam::123456789012:role/demo"
 -DurationInSeconds 3600 -ExternalId "ABC123"
```

- For API details, see AssumeRole in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Assume an IAM role that requires an MFA token and use temporary credentials to list Amazon S3 buckets for the account.

```
def list_buckets_from_assumed_role_with_mfa(
    assume_role_arn, session_name, mfa_serial_number, mfa_totp, sts_client
):
    """
    Assumes a role from another account and uses the temporary credentials from
    that role to list the Amazon S3 buckets that are owned by the other account.
    Requires an MFA device serial number and token.

    The assumed role must grant permission to list the buckets in the other
 account.

    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                            grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    :param mfa_serial_number: The serial number of the MFA device. For a virtual
 MFA
                              device, this is an ARN.
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the
 role.
    """
    response = sts_client.assume_role(
        RoleArn=assume_role_arn,
```

```
        RoleSessionName=session_name,
        SerialNumber=mfa_serial_number,
        TokenCode=mfa_totp,
    )
    temp_credentials = response["Credentials"]
    print(f"Assumed role {assume_role_arn} and got temporary credentials.")

    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )

    print(f"Listing buckets for the assumed role's account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
```

- For API details, see AssumeRole in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
 # Creates an AWS Security Token Service (AWS STS) client with specified
credentials.
 # This is separated into a factory function so that it can be mocked for unit
testing.
 #
 # @param key_id [String] The ID of the access key used by the STS client.
 # @param key_secret [String] The secret part of the access key used by the STS
client.
 def create_sts_client(key_id, key_secret)
```

```
    Aws::STS::Client.new(access_key_id: key_id, secret_access_key: key_secret)
  end

  # Gets temporary credentials that can be used to assume a role.
  #
  # @param role_arn [String] The ARN of the role that is assumed when these
credentials
  #                          are used.
  # @param sts_client [AWS::STS::Client] An AWS STS client.
  # @return [Aws::AssumeRoleCredentials] The credentials that can be used to
assume the role.
  def assume_role(role_arn, sts_client)
    credentials = Aws::AssumeRoleCredentials.new(
      client: sts_client,
      role_arn: role_arn,
      role_session_name: 'create-use-assume-role-scenario'
    )
    @logger.info("Assumed role '#{role_arn}', got temporary credentials.")
    credentials
  end
```

- For API details, see AssumeRole in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn assume_role(config: &SdkConfig, role_name: String, session_name:
 Option<String>) {
    let provider = aws_config::sts::AssumeRoleProvider::builder(role_name)
        .session_name(session_name.unwrap_or("rust_sdk_example_session".into()))
        .configure(config)
        .build()
        .await;
```

```
    let local_config = aws_config::from_env()
        .credentials_provider(provider)
        .load()
        .await;
    let client = Client::new(&local_config);
    let req = client.get_caller_identity();
    let resp = req.send().await;
    match resp {
        Ok(e) => {
            println!("UserID :               {}",
 e.user_id().unwrap_or_default());
            println!("Account:               {}",
 e.account().unwrap_or_default());
            println!("Arn    :               {}", e.arn().unwrap_or_default());
        }
        Err(e) => println!("{:?}", e),
    }
}
```

- For API details, see AssumeRole in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSSTS

    public func assumeRole(role: IAMClientTypes.Role, sessionName: String)
        async throws -> STSClientTypes.Credentials
    {
        let input = AssumeRoleInput(
            roleArn: role.arn,
            roleSessionName: sessionName
        )
        do {
```

```
            let output = try await stsClient.assumeRole(input: input)

            guard let credentials = output.credentials else {
                throw ServiceHandlerError.authError
            }

            return credentials
        } catch {
            print("Error assuming role: ", dump(error))
            throw error
        }
    }
```

- For API details, see AssumeRole in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AssumeRoleWithWebIdentity` with a CLI

The following code examples show how to use `AssumeRoleWithWebIdentity`.

CLI

AWS CLI

To get short-term credentials for a role authenticated with Web Identity (OAuth 2."0)

The following `assume-role-with-web-identity` example retrieves a set of short-term credentials for the IAM role `app1`. The request is authenticated by using the web identity token supplied by the specified web identity provider. Two additional policies are applied to the session to further restrict what the user can do. The returned credentials expire one hour after they are generated.

```
aws sts assume-role-with-web-identity \
    --duration-seconds 3600 \
    --role-session-name "app1" \
    --provider-id "www.amazon.com" \
```

```
    --policy-arns arn=arn:aws:iam::123456789012:policy/
webidentitydemopolicy1 arn=arn:aws:iam::123456789012:policy/
webidentitydemopolicy2 \
    --role-arn arn:aws:iam::123456789012:role/FederatedWebIdentityRole \
    --web-identity-token "Atza
%7CIQEBLjAsAhRFiXuWpUXuRvQ9PZL3GMFcYevydwIUFAHZwXZXXXXXXXXJnrulxKDHwy87oGKPznh0D6bEQZTSCz
CrKqjG7nPBjNIL016GGvuS5gSvPRUxWES3VYfm1wl7WTI7jn-Pcb6M-
buCgHhFOzTQxod27L9CqnOLio7N3gZAGpsp6n1-
AJBOCJckcyXe2c6uD0srOJeZlKUm2eTDVMf8IehDVI0r1QOnTV6KzzAI3OY87Vd_cVMQ"
```

Output:

```
{
    "SubjectFromWebIdentityToken": "amzn1.account.AF6RHO7KZU5XRVQJGXK6HB56KR2A",
    "Audience": "client.5498841531868486423.1548@apps.example.com",
    "AssumedRoleUser": {
        "Arn": "arn:aws:sts::123456789012:assumed-role/FederatedWebIdentityRole/
app1",
        "AssumedRoleId": "AROACLKWSDQRAOEXAMPLE:app1"
    },
    "Credentials": {
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY",
        "SessionToken": "AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4OlgkBN9bkUDNCJiBeb/
AXlzBBko7b15fjrBs2+cTQtpZ3CYWFXG8C5zqx37wnOE49mRl/+OtkIKGO7fAE",
        "Expiration": "2020-05-19T18:06:10+00:00"
    },
    "Provider": "www.amazon.com"
}
```

For more information, see Requesting Temporary Security Credentials in the *AWS IAM User Guide*.

- For API details, see AssumeRoleWithWebIdentity in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns a temporary set of credentials, valid for one hour, for a user who has been authenticated with the Login with Amazon identity provider. The credentials

assume the access policy associated with the role identified by the role ARN. Optionally, you can pass a JSON policy to the -Policy parameter that further refine the access permissions (you cannot grant more permissions than are available in the permissions associated with the role). The value supplied to the -WebIdentityToken is the unique user identifier that was returned by the identity provider.

```
Use-STSWebIdentityRole -DurationInSeconds 3600 -ProviderId "www.amazon.com"
 -RoleSessionName "app1" -RoleArn "arn:aws:iam::123456789012:role/
FederatedWebIdentityRole" -WebIdentityToken "Atza...DVI0r1"
```

- For API details, see AssumeRoleWithWebIdentity in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns a temporary set of credentials, valid for one hour, for a user who has been authenticated with the Login with Amazon identity provider. The credentials assume the access policy associated with the role identified by the role ARN. Optionally, you can pass a JSON policy to the -Policy parameter that further refine the access permissions (you cannot grant more permissions than are available in the permissions associated with the role). The value supplied to the -WebIdentityToken is the unique user identifier that was returned by the identity provider.

```
Use-STSWebIdentityRole -DurationInSeconds 3600 -ProviderId "www.amazon.com"
 -RoleSessionName "app1" -RoleArn "arn:aws:iam::123456789012:role/
FederatedWebIdentityRole" -WebIdentityToken "Atza...DVI0r1"
```

- For API details, see AssumeRoleWithWebIdentity in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DecodeAuthorizationMessage` with a CLI

The following code examples show how to use `DecodeAuthorizationMessage`.

CLI

AWS CLI

To decode an encoded authorization message returned in response to a request

The following `decode-authorization-message` example decodes additional information about the authorization status of a request from an encoded message returned in response to an Amazon Web Services request.

```
aws sts decode-authorization-message \
    --encoded-message EXAMPLEWodyRNrtlQARDip-
eTA6i6DrlUhHhPQrLWB_lAbl5pAKxl9mPDLexYcGBreyIKQC1BGBIpBKr3dFDkwqeO7e2NMk5j_hmzAiChJN-8oy3
Ojau7BMjOTWwOtHPhV_Zaz87yENdipr745EjQwRd5LaoL3vN8_5ZfA9UiBMKDgVh1gjqZJFUiQoubv78V1RbHNYnK
pOu3FZjwYStfvTb3GHs3-6rLribGO9jZOktkfE6vqxlFzLyeDr4P2ihC1wty9tArCvvGzIAUNmARQJ2VVWPxioqgo
JWP5pwe_mAyqh0NLw-r1S56YC_90onj9A80sNrHlI-
tIiNd7tgNTYzDuPQYD2FMDBnp82V9eVmYGtPp5NIeSpuf3fOHanFuBZgENxZQZ2dlH3xJGMTtYayzZrRXjiq_SfX9
FaoPIb8LmmKVBLpIB0iFhU9sEHPqKHVPi6jdxXqKaZaFGvYVmVOiuQdNQKuyk0p067POFrZECLjjOtNPBOZCcuEKE
```

Output:

```
{
    "DecodedMessage": "{\"allowed\":false,\"explicitDeny\":true,
\"matchedStatements\":{\"items\":[{\"statementId\":\"VisualEditor0\",\"effect
\":\"DENY\",\"principals\":{\"items\":[{\"value\":\"AROA123456789EXAMPLE
\"}]},\"principalGroups\":{\"items\":[]},\"actions\":{\"items\":[{\"value
\":\"ec2:RunInstances\"}]},\"resources\":{\"items\":[{\"value\":\"*
\"}]},\"conditions\":{\"items\":[]}}]},\"failures\":{\"items\":[]},
\"context\":{\"principal\":{\"id\":\"AROA123456789EXAMPLE:Ana\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Developer/Ana\"},\"action\":
\"RunInstances\",\"resource\":\"arn:aws:ec2:us-west-2:123456789012:instance/*
\",\"conditions\":{\"items\":[{\"key\":\"ec2:MetadataHttpPutResponseHopLimit\",
\"values\":{\"items\":[{\"value\":\"2\"}]}},{\"key\":\"ec2:InstanceMarketType
\",\"values\":{\"items\":[{\"value\":\"on-demand\"}]}},{\"key\":\"aws:Resource
\",\"values\":{\"items\":[{\"value\":\"instance/*\"}]}},{\"key\":\"aws:Account
\",\"values\":{\"items\":[{\"value\":\"123456789012\"}]}},{\"key\":
\"ec2:AvailabilityZone\",\"values\":{\"items\":[{\"value\":\"us-west-2a\"}]}},
{\"key\":\"ec2:ebsOptimized\",\"values\":{\"items\":[{\"value\":\"false\"}]}},
{\"key\":\"ec2:IsLaunchTemplateResource\",\"values\":{\"items\":[{\"value\":
\"false\"}]}},{\"key\":\"ec2:InstanceType\",\"values\":{\"items\":[{\"value
\":\"t2.micro\"}]}},{\"key\":\"ec2:RootDeviceType\",\"values\":{\"items\":
[{\"value\":\"ebs\"}]}},{\"key\":\"aws:Region\",\"values\":{\"items\":[{\"value
\":\"us-west-2\"}]}},{\"key\":\"ec2:MetadataHttpEndpoint\",\"values\":{\"items
```

```
\":[{\"value\":\"enabled\"}]}},{\"key\":\"aws:Service\",\"values\":{\"items
\":[{\"value\":\"ec2\"}]}},{\"key\":\"ec2:InstanceID\",\"values\":{\"items\":
[{\"value\":\"*\"}]}},{\"key\":\"ec2:MetadataHttpTokens\",\"values\":{\"items
\":[{\"value\":\"required\"}]}},{\"key\":\"aws:Type\",\"values\":{\"items
\":[{\"value\":\"instance\"}]}},{\"key\":\"ec2:Tenancy\",\"values\":{\"items
\":[{\"value\":\"default\"}]}},{\"key\":\"ec2:Region\",\"values\":{\"items
\":[{\"value\":\"us-west-2\"}]}},{\"key\":\"aws:ARN\",\"values\":{\"items\":
[{\"value\":\"arn:aws:ec2:us-west-2:123456789012:instance/*\"}]}}]}}}"
}
```

For more information, see Policy evaluation logic in the *AWS IAM User Guide*.

- For API details, see DecodeAuthorizationMessage in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Decodes the additional information contained in the supplied encoded message content that was returned in response to a request. The additional information is encoded because details of the authorization status can constitute privileged information that the user who requested the action should not see.

```
Convert-STSAuthorizationMessage -EncodedMessage "...encoded message..."
```

- For API details, see DecodeAuthorizationMessage in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Decodes the additional information contained in the supplied encoded message content that was returned in response to a request. The additional information is encoded because details of the authorization status can constitute privileged information that the user who requested the action should not see.

```
Convert-STSAuthorizationMessage -EncodedMessage "...encoded message..."
```

- For API details, see DecodeAuthorizationMessage in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetFederationToken` with a CLI

The following code examples show how to use `GetFederationToken`.

CLI

AWS CLI

To return a set of temporary security credentials using IAM user access key credentials

The following `get-federation-token` example returns a set of temporary security credentials (consisting of an access key ID, a secret access key, and a security token) for a user. You must call the `GetFederationToken` operation using the long-term security credentials of an IAM user.

```
aws sts get-federation-token \
    --name Bob \
    --policy file://myfile.json \
    --policy-arns arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
    --duration-seconds 900
```

Contents of `myfile.json`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:Describe*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "elasticloadbalancing:Describe*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:Describe*",
            "Resource": "*"
        }
    ]
}
```

Output:

```
{
    "Credentials": {
        "AccessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": "EXAMPLEpZ2luX2VjEGoaCXVzLXdlc3QtMiJIMEYCIQC/
W9pL5ArQyDD5JwFL3/h5+WGopQ24GEXweNctwhi9sgIhAMkg
+MZE35iWM8s4r5Lr25f9rSTVPFH98G42QQunWMTfKq0DCOP/////////
wEQAxoMNDUyOTI1MTcwNTA3Igxuy3AOpuuoLsk3MJwqgQPg8QOd9HuoClUxq26wnc/nm
+eZLjHDyGf2KUAHK2DuaS/nrGSEXAMPLE",
        "Expiration": "2023-12-20T02:06:07+00:00"
    },
    "FederatedUser": {
        "FederatedUserId": "123456789012:Bob",
        "Arn": "arn:aws:sts::123456789012:federated-user/Bob"
    },
    "PackedPolicySize": 36
}
```

For more information, see Requesting Temporary Security Credentials in the *AWS IAM User Guide*.

- For API details, see GetFederationToken in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Requests a federated token valid for one hour using "Bob" as the name of the federated user. This name can be used to reference the federated user name in a resource-based policy (such as an Amazon S3 bucket policy). The supplied IAM policy, in JSON format, is used to scope down the permissions that are available to the IAM user. The supplied policy cannot grant more permissions than those granted to the requesting user, with the final permissions for the federated user being the most restrictive set based on the intersection of the passed policy and the IAM user policy.

```
Get-STSFederationToken -Name "Bob" -Policy "...JSON policy..." -DurationInSeconds
 3600
```

- For API details, see GetFederationToken in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Requests a federated token valid for one hour using "Bob" as the name of the federated user. This name can be used to reference the federated user name in a resource-based policy (such as an Amazon S3 bucket policy). The supplied IAM policy, in JSON format, is used to scope down the permissions that are available to the IAM user. The supplied policy cannot grant more permissions than those granted to the requesting user, with the final permissions for the federated user being the most restrictive set based on the intersection of the passed policy and the IAM user policy.

```
Get-STSFederationToken -Name "Bob" -Policy "...JSON policy..." -DurationInSeconds
 3600
```

- For API details, see GetFederationToken in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetSessionToken` with an AWS SDK or CLI

The following code examples show how to use `GetSessionToken`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get a session token that requires an MFA token

CLI

AWS CLI

To get a set of short term credentials for an IAM identity

The following `get-session-token` example retrieves a set of short-term credentials for the IAM identity making the call. The resulting credentials can be used for requests where multi-factor authentication (MFA) is required by policy. The credentials expire 15 minutes after they are generated.

```
aws sts get-session-token \
    --duration-seconds 900 \
    --serial-number "YourMFADeviceSerialNumber" \
    --token-code 123456
```

Output:

```
{
    "Credentials": {
        "AccessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY",
        "SessionToken": "AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4OlgkBN9bkUDNCJiBeb/
AXlzBBko7b15fjrBs2+cTQtpZ3CYWFXG8C5zqx37wnOE49mRl/+OtkIKGO7fAE",
        "Expiration": "2020-05-19T18:06:10+00:00"
    }
}
```

For more information, see Requesting Temporary Security Credentials in the *AWS IAM User Guide*.

- For API details, see GetSessionToken in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for a set period of time. The credentials used to request temporary credentials are inferred from the current shell defaults. To specify other credentials, use the -ProfileName or -AccessKey/-SecretKey parameters.

```
Get-STSSessionToken
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

Example 2: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour. The credentials used to make the request are obtained from the specified profile.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

Example 3: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour using the identification number of the MFA device associated with the account whose credentials are specified in the profile 'myprofilename' and the value provided by the device.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile -SerialNumber
 YourMFADeviceSerialNumber -TokenCode 123456
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

- For API details, see GetSessionToken in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for a set period of time. The credentials used to request temporary credentials are inferred from the current shell defaults. To specify other credentials, use the -ProfileName or -AccessKey/-SecretKey parameters.

```
Get-STSSessionToken
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

Example 2: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour. The credentials used to make the request are obtained from the specified profile.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

Example 3: Returns an `Amazon.RuntimeAWSCredentials` instance containing temporary credentials valid for one hour using the identification number of the MFA device associated with the account whose credentials are specified in the profile 'myprofilename' and the value provided by the device.

```
Get-STSSessionToken -DurationInSeconds 3600 -ProfileName myprofile -SerialNumber
 YourMFADeviceSerialNumber -TokenCode 123456
```

Output:

```
AccessKeyId                              Expiration
 SecretAccessKey                         SessionToken
-----------                              ----------
 ---------------                         ------------
EXAMPLEACCESSKEYID                       2/16/2015 9:12:28 PM
 examplesecretaccesskey...               SamPleTokeN.....
```

- For API details, see GetSessionToken in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get a session token by passing an MFA token and use it to list Amazon S3 buckets for the account.

```
def list_buckets_with_session_token_with_mfa(mfa_serial_number, mfa_totp,
 sts_client):
    """
    Gets a session token with MFA credentials and uses the temporary session
    credentials to list Amazon S3 buckets.

    Requires an MFA device serial number and token.

    :param mfa_serial_number: The serial number of the MFA device. For a virtual
 MFA
                              device, this is an Amazon Resource Name (ARN).
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the
 role.
    """
    if mfa_serial_number is not None:
        response = sts_client.get_session_token(
            SerialNumber=mfa_serial_number, TokenCode=mfa_totp
        )
    else:
        response = sts_client.get_session_token()
    temp_credentials = response["Credentials"]

    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )

    print(f"Buckets for the account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
```

- For API details, see GetSessionToken in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for AWS STS using AWS SDKs

The following code examples show you how to implement common scenarios in AWS STS with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within AWS STS or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK
- Configure container service connectivity
- Construct a URL with AWS STS for federated users using an AWS SDK
- Create a container task for the serverless launch type
- Create a rest API with function proxy integration
- Creating a container service for virtual machine instances
- Creating a managed monitoring workspace
- Creating a monitoring dashboard with function name as a variable
- Get a session token that requires an MFA token with AWS STS using an AWS SDK
- Getting started with configuration management
- Getting started with container registries
- Getting started with internet of things messaging
- Getting started with machine learning feature stores
- Getting started with managed kubernetes clusters
- Getting started with managed streaming
- Getting started with push notifications
- Getting started with search and analytics engines
- Getting started with web application firewalls
- Moving hardcoded secrets to secure secret storage
- Run CPU stress tests on virtual machine instances using fault injection
- Setting up systems management

Assume an IAM role that requires an MFA token with AWS STS using an AWS SDK

The following code example shows how to assume a role that requires an MFA token.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create an IAM role that grants permission to list Amazon S3 buckets.
- Create an IAM user that has permission to assume the role only when MFA credentials are provided.
- Register an MFA device for the user.
- Assume the role and use temporary credentials to list S3 buckets.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user, register an MFA device, and create a role that grants permission to list S3 buckets. The user has rights only to assume the role.

```
def setup(iam_resource):
    """
    Creates a new user with no permissions.
    Creates a new virtual MFA device.
    Displays the QR code to seed the device.
    Asks for two codes from the MFA device.
    Registers the MFA device for the user.
    Creates an access key pair for the user.
```

```
    Creates a role with a policy that lets the user assume the role and requires
MFA.
    Creates a policy that allows listing Amazon S3 buckets.
    Attaches the policy to the role.
    Creates an inline policy for the user that lets the user assume the role.

    For demonstration purposes, the user is created in the same account as the
role,
    but in practice the user would likely be from another account.

    Any MFA device that can scan a QR code will work with this demonstration.
    Common choices are mobile apps like LastPass Authenticator,
    Microsoft Authenticator, or Google Authenticator.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM)
resource
                         that has permissions to create users, roles, and
policies
                         in the account.
    :return: The newly created user, user key, virtual MFA device, and role.
    """
    user = iam_resource.create_user(UserName=unique_name("user"))
    print(f"Created user {user.name}.")

    virtual_mfa_device = iam_resource.create_virtual_mfa_device(
        VirtualMFADeviceName=unique_name("mfa")
    )
    print(f"Created virtual MFA device {virtual_mfa_device.serial_number}")

    print(
        f"Showing the QR code for the device. Scan this in the MFA app of your "
        f"choice."
    )
    with open("qr.png", "wb") as qr_file:
        qr_file.write(virtual_mfa_device.qr_code_png)
    webbrowser.open(qr_file.name)

    print(f"Enter two consecutive code from your MFA device.")
    mfa_code_1 = input("Enter the first code: ")
    mfa_code_2 = input("Enter the second code: ")
    user.enable_mfa(
        SerialNumber=virtual_mfa_device.serial_number,
        AuthenticationCode1=mfa_code_1,
        AuthenticationCode2=mfa_code_2,
```

```
    )
    os.remove(qr_file.name)
    print(f"MFA device is registered with the user.")

    user_key = user.create_access_key_pair()
    print(f"Created access key pair for user.")

    print(f"Wait for user to be ready.", end="")
    progress_bar(10)

    role = iam_resource.create_role(
        RoleName=unique_name("role"),
        AssumeRolePolicyDocument=json.dumps(
            {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {"AWS": user.arn},
                        "Action": "sts:AssumeRole",
                        "Condition": {"Bool": {"aws:MultiFactorAuthPresent":
True}},
                    }
                ],
            }
        ),
    )
    print(f"Created role {role.name} that requires MFA.")

    policy = iam_resource.create_policy(
        PolicyName=unique_name("policy"),
        PolicyDocument=json.dumps(
            {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Action": "s3:ListAllMyBuckets",
                        "Resource": "arn:aws:s3:::*",
                    }
                ],
            }
        ),
    )
```

```
    role.attach_policy(PolicyArn=policy.arn)
    print(f"Created policy {policy.policy_name} and attached it to the role.")

    user.create_policy(
        PolicyName=unique_name("user-policy"),
        PolicyDocument=json.dumps(
            {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Action": "sts:AssumeRole",
                        "Resource": role.arn,
                    }
                ],
            }
        ),
    )
    print(
        f"Created an inline policy for {user.name} that lets the user assume "
        f"the role."
    )

    print("Give AWS time to propagate these new resources and connections.",
 end="")
    progress_bar(10)

    return user, user_key, virtual_mfa_device, role
```

Show that assuming the role without an MFA token is not allowed.

```
def try_to_assume_role_without_mfa(assume_role_arn, session_name, sts_client):
    """
    Shows that attempting to assume the role without sending MFA credentials
 results
    in an AccessDenied error.

    :param assume_role_arn: The Amazon Resource Name (ARN) of the role to assume.
    :param session_name: The name of the STS session.
```

```
    :param sts_client: A Boto3 STS instance that has permission to assume the
 role.
    """
    print(f"Trying to assume the role without sending MFA credentials...")
    try:
        sts_client.assume_role(RoleArn=assume_role_arn,
 RoleSessionName=session_name)
        raise RuntimeError("Expected AccessDenied error.")
    except ClientError as error:
        if error.response["Error"]["Code"] == "AccessDenied":
            print("Got AccessDenied.")
        else:
            raise
```

Assume the role that grants permission to list S3 buckets, passing the required MFA token, and show that buckets can be listed.

```
def list_buckets_from_assumed_role_with_mfa(
    assume_role_arn, session_name, mfa_serial_number, mfa_totp, sts_client
):
    """
    Assumes a role from another account and uses the temporary credentials from
    that role to list the Amazon S3 buckets that are owned by the other account.
    Requires an MFA device serial number and token.

    The assumed role must grant permission to list the buckets in the other
 account.

    :param assume_role_arn: The Amazon Resource Name (ARN) of the role that
                            grants access to list the other account's buckets.
    :param session_name: The name of the STS session.
    :param mfa_serial_number: The serial number of the MFA device. For a virtual
 MFA
                              device, this is an ARN.
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the
 role.
    """
    response = sts_client.assume_role(
        RoleArn=assume_role_arn,
```

```
        RoleSessionName=session_name,
        SerialNumber=mfa_serial_number,
        TokenCode=mfa_totp,
    )
    temp_credentials = response["Credentials"]
    print(f"Assumed role {assume_role_arn} and got temporary credentials.")

    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )

    print(f"Listing buckets for the assumed role's account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
```

Destroy the resources created for the demo.

```
def teardown(user, virtual_mfa_device, role):
    """
    Removes all resources created during setup.

    :param user: The demo user.
    :param role: The demo role.
    """
    for attached in role.attached_policies.all():
        policy_name = attached.policy_name
        role.detach_policy(PolicyArn=attached.arn)
        attached.delete()
        print(f"Detached and deleted {policy_name}.")
    role.delete()
    print(f"Deleted {role.name}.")
    for user_pol in user.policies.all():
        user_pol.delete()
        print("Deleted inline user policy.")
    for key in user.access_keys.all():
        key.delete()
        print("Deleted user's access key.")
```

```
    for mfa in user.mfa_devices.all():
        mfa.disassociate()
    virtual_mfa_device.delete()
    user.delete()
    print(f"Deleted {user.name}.")
```

Run this scenario by using the previously defined functions.

```
def usage_demo():
    """Drives the demonstration."""
    print("-" * 88)
    print(
        f"Welcome to the AWS Security Token Service assume role demo, "
        f"starring multi-factor authentication (MFA)!"
    )
    print("-" * 88)
    iam_resource = boto3.resource("iam")
    user, user_key, virtual_mfa_device, role = setup(iam_resource)
    print(f"Created {user.name} and {role.name}.")
    try:
        sts_client = boto3.client(
            "sts", aws_access_key_id=user_key.id,
 aws_secret_access_key=user_key.secret
        )
        try_to_assume_role_without_mfa(role.arn, "demo-sts-session", sts_client)
        mfa_totp = input("Enter the code from your registered MFA device: ")
        list_buckets_from_assumed_role_with_mfa(
            role.arn,
            "demo-sts-session",
            virtual_mfa_device.serial_number,
            mfa_totp,
            sts_client,
        )
    finally:
        teardown(user, virtual_mfa_device, role)
        print("Thanks for watching!")
```

- For API details, see AssumeRole in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Configure container service connectivity

The following code example shows how to:

- Create the VPC infrastructure
- Set up logging
- Create the ECS cluster
- Configure IAM roles
- Create the service with Service Connect
- Verify the deployment
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# ECS Service Connect Tutorial Script v4 - Modified to use Default VPC
# This script creates an ECS cluster with Service Connect and deploys an nginx
 service
# Uses the default VPC to avoid VPC limits

set -e  # Exit on any error

# Configuration
```

```
SCRIPT_NAME="ECS Service Connect Tutorial"
LOG_FILE="ecs-service-connect-tutorial-v4-default-vpc.log"
REGION=${AWS_DEFAULT_REGION:-${AWS_REGION:-$(aws configure get region 2>/dev/
null)}}
if [ -z "$REGION" ]; then
    echo "ERROR: No AWS region configured."
    echo "Set one with: aws configure set region us-east-1"
    exit 1
fi
ENV_PREFIX="tutorial"
CLUSTER_NAME="${ENV_PREFIX}-cluster"
NAMESPACE_NAME="service-connect"

# Generate random suffix for unique resource names
RANDOM_SUFFIX=$(openssl rand -hex 6)

# Arrays to track created resources for cleanup
declare -a CREATED_RESOURCES=()

# Logging function
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

# Error handling function
handle_error() {
    log "ERROR: Script failed at line $1"
    log "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Set up error handling
trap 'handle_error $LINENO' ERR

# Function to add resource to tracking array
track_resource() {
    CREATED_RESOURCES+=("$1")
    log "Tracking resource: $1"
}

# Function to check if command output contains actual errors
check_for_errors() {
    local output="$1"
```

```
    local command_name="$2"

    # Check for specific AWS CLI error patterns, not just any occurrence of
 "error"
    if echo "$output" | grep -qi "An error occurred\|InvalidParameterException\|
AccessDenied\|ValidationException\|ResourceNotFoundException"; then
        log "ERROR in $command_name: $output"
        return 1
    fi
    return 0
}

# Function to get AWS account ID
get_account_id() {
    ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
    log "Using AWS Account ID: $ACCOUNT_ID"
}

# Function to wait for resources to be ready
wait_for_resource() {
    local resource_type="$1"
    local resource_id="$2"

    case "$resource_type" in
        "cluster")
            log "Waiting for cluster $resource_id to be active..."
            local attempt=1
            local max_attempts=30
            while [ $attempt -le $max_attempts ]; do
                local status=$(aws ecs describe-clusters --clusters
 "$resource_id" --query 'clusters[0].status' --output text)
                if [ "$status" = "ACTIVE" ]; then
                    log "Cluster is now active"
                    return 0
                fi
                log "Cluster status: $status (attempt $attempt/$max_attempts)"
                sleep 10
                ((attempt++))
            done
            log "ERROR: Cluster did not become active within expected time"
            return 1
            ;;
        "service")
            log "Waiting for service $resource_id to be stable..."
```

```
            aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
 "$resource_id"
            ;;
        "nat-gateway")
            log "Waiting for NAT Gateway $resource_id to be available..."
            aws ec2 wait nat-gateway-available --nat-gateway-ids "$resource_id"
            ;;
    esac
}

# Function to use default VPC infrastructure
setup_default_vpc_infrastructure() {
    log "Using default VPC infrastructure..."

    # Get default VPC
    VPC_ID=$(aws ec2 describe-vpcs --filters "Name=isDefault,Values=true" --query
 'Vpcs[0].VpcId' --output text)
    if [[ "$VPC_ID" == "None" || -z "$VPC_ID" ]]; then
        log "ERROR: No default VPC found. Please create a default VPC first."
        exit 1
    fi
    log "Using default VPC: $VPC_ID"

    # Get default subnets
    SUBNETS=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=$VPC_ID"
 "Name=default-for-az,Values=true" --query 'Subnets[].SubnetId' --output text)
    SUBNET_ARRAY=($SUBNETS)

    if [ ${#SUBNET_ARRAY[@]} -lt 2 ]; then
        log "ERROR: Need at least 2 subnets for ECS Service Connect. Found:
 ${#SUBNET_ARRAY[@]}"
        exit 1
    fi

    PUBLIC_SUBNET1=${SUBNET_ARRAY[0]}
    PUBLIC_SUBNET2=${SUBNET_ARRAY[1]}

    log "Using subnets: $PUBLIC_SUBNET1, $PUBLIC_SUBNET2"

    # Create security group for ECS tasks
    SG_OUTPUT=$(aws ec2 create-security-group \
        --group-name "${ENV_PREFIX}-ecs-sg-${RANDOM_SUFFIX}" \
        --description "Security group for ECS Service Connect tutorial" \
        --vpc-id "$VPC_ID" \
```

```
        --tag-specifications 'ResourceType=security-
group,Tags=[{Key=project,Value=doc-smith},{Key=tutorial,Value=amazon-ecs-service-
connect}]' 2>&1)
    check_for_errors "$SG_OUTPUT" "create-security-group"
    SECURITY_GROUP_ID=$(echo "$SG_OUTPUT" | grep -o '"GroupId": "[^"]*"' | cut -
d'"' -f4)
    track_resource "SG:$SECURITY_GROUP_ID"
    log "Created security group: $SECURITY_GROUP_ID"

    # Add inbound rules to security group
    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
        --protocol tcp \
        --port 80 \
        --cidr 0.0.0.0/0 >/dev/null 2>&1 || true

    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
        --protocol tcp \
        --port 443 \
        --cidr 0.0.0.0/0 >/dev/null 2>&1 || true

    log "Default VPC infrastructure setup completed"
}

# Function to create CloudWatch log groups
create_log_groups() {
    log "Creating CloudWatch log groups..."

    # Create log group for nginx container
    aws logs create-log-group --log-group-name "/ecs/service-connect-nginx"
 --tags project=doc-smith,tutorial=amazon-ecs-service-connect 2>&1 | grep -v
 "ResourceAlreadyExistsException" || {
        if [ ${PIPESTATUS[0]} -eq 0 ]; then
            log "Log group /ecs/service-connect-nginx created"
            track_resource "LOG_GROUP:/ecs/service-connect-nginx"
        else
            log "Log group /ecs/service-connect-nginx already exists"
        fi
    }

    # Create log group for service connect proxy
```

```
    aws logs create-log-group --log-group-name "/ecs/service-connect-proxy"
 --tags project=doc-smith,tutorial=amazon-ecs-service-connect 2>&1 | grep -v
 "ResourceAlreadyExistsException" || {
        if [ ${PIPESTATUS[0]} -eq 0 ]; then
            log "Log group /ecs/service-connect-proxy created"
            track_resource "LOG_GROUP:/ecs/service-connect-proxy"
        else
            log "Log group /ecs/service-connect-proxy already exists"
        fi
    }
}

# Function to create ECS cluster with Service Connect
create_ecs_cluster() {
    log "Creating ECS cluster with Service Connect..."

    CLUSTER_OUTPUT=$(aws ecs create-cluster \
        --cluster-name "$CLUSTER_NAME" \
        --service-connect-defaults namespace="$NAMESPACE_NAME" \
        --tags key=Environment,value=tutorial key=project,value=doc-smith
 key=tutorial,value=amazon-ecs-service-connect 2>&1)
    check_for_errors "$CLUSTER_OUTPUT" "create-cluster"

    track_resource "CLUSTER:$CLUSTER_NAME"
    log "Created ECS cluster: $CLUSTER_NAME"

    wait_for_resource "cluster" "$CLUSTER_NAME"

    # Track the Service Connect namespace that gets created
    # Wait a moment for the namespace to be created
    sleep 5
    NAMESPACE_ID=$(aws servicediscovery list-namespaces \
        --filters Name=TYPE,Values=HTTP \
        --query "Namespaces[?Name=='$NAMESPACE_NAME'].Id" --output text 2>/dev/
null || echo "")

    if [[ -n "$NAMESPACE_ID" && "$NAMESPACE_ID" != "None" ]]; then
        track_resource "NAMESPACE:$NAMESPACE_ID"
        log "Service Connect namespace created: $NAMESPACE_ID"
    fi
}

# Function to create IAM roles
create_iam_roles() {
```

```
    log "Creating IAM roles..."

    # Check if ecsTaskExecutionRole exists
    if aws iam get-role --role-name ecsTaskExecutionRole >/dev/null 2>&1; then
        log "IAM role ecsTaskExecutionRole exists"
    else
        log "Creating ecsTaskExecutionRole..."
        aws iam create-role \
            --role-name ecsTaskExecutionRole \
            --assume-role-policy-document '{
                "Version":"2012-10-17",
                "Statement": [{
                    "Effect": "Allow",
                    "Principal": {"Service": "ecs-tasks.amazonaws.com"},
                    "Action": "sts:AssumeRole"
                }]
            }' >/dev/null 2>&1
        aws iam attach-role-policy \
            --role-name ecsTaskExecutionRole \
            --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy >/dev/null 2>&1
        track_resource "ROLE:ecsTaskExecutionRole"
        aws iam tag-role --role-name ecsTaskExecutionRole --tags
 Key=project,Value=doc-smith Key=tutorial,Value=amazon-ecs-service-connect
        log "Created ecsTaskExecutionRole"
        sleep 10
    fi

    # Check if ecsTaskRole exists, create if not
    if aws iam get-role --role-name ecsTaskRole >/dev/null 2>&1; then
        log "IAM role ecsTaskRole exists"
    else
        log "IAM role ecsTaskRole does not exist, will create it"

        # Create trust policy for ECS tasks
        cat > /tmp/ecs-task-trust-policy.json << EOF
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
```

```
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF

        aws iam create-role \
            --role-name ecsTaskRole \
            --assume-role-policy-document file:///tmp/ecs-task-trust-policy.json
 >/dev/null

        track_resource "IAM_ROLE:ecsTaskRole"
        aws iam tag-role --role-name ecsTaskRole --tags Key=project,Value=doc-
smith Key=tutorial,Value=amazon-ecs-service-connect
        log "Created ecsTaskRole"

        # Wait for role to be available
        sleep 10
    fi
}

# Function to create task definition
create_task_definition() {
    log "Creating task definition..."

    # Create task definition JSON
    cat > /tmp/task-definition.json << EOF
{
    "family": "service-connect-nginx",
    "networkMode": "awsvpc",
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "256",
    "memory": "512",
    "executionRoleArn": "arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskExecutionRole",
    "taskRoleArn": "arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskRole",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/docker/library/nginx:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "protocol": "tcp",
                    "name": "nginx-port"
```

```
                }
            ],
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/service-connect-nginx",
                    "awslogs-region": "${REGION}",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ]
}
EOF

    TASK_DEF_OUTPUT=$(aws ecs register-task-definition --cli-input-json file:///
tmp/task-definition.json 2>&1)
    check_for_errors "$TASK_DEF_OUTPUT" "register-task-definition"

    TASK_DEF_ARN=$(echo "$TASK_DEF_OUTPUT" | grep -o '"taskDefinitionArn":
 "[^"]*"' | cut -d'"' -f4)
    track_resource "TASK_DEF:service-connect-nginx"
    log "Created task definition: $TASK_DEF_ARN"

    # Clean up temporary file
    rm -f /tmp/task-definition.json
}

# Function to create ECS service with Service Connect
create_ecs_service() {
    log "Creating ECS service with Service Connect..."

    # Create service definition JSON
    cat > /tmp/service-definition.json << EOF
{
    "serviceName": "service-connect-nginx-service",
    "cluster": "${CLUSTER_NAME}",
    "taskDefinition": "service-connect-nginx",
    "desiredCount": 1,
    "launchType": "FARGATE",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": ["${PUBLIC_SUBNET1}", "${PUBLIC_SUBNET2}"],
```

```
            "securityGroups": ["${SECURITY_GROUP_ID}"],
            "assignPublicIp": "ENABLED"
        }
    },
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "${NAMESPACE_NAME}",
        "services": [
            {
                "portName": "nginx-port",
                "discoveryName": "nginx",
                "clientAliases": [
                    {
                        "port": 80,
                        "dnsName": "nginx"
                    }
                ]
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "${REGION}",
                "awslogs-stream-prefix": "ecs-service-connect"
            }
        }
    },
    "tags": [
        {
            "key": "Environment",
            "value": "tutorial"
        },
        {
            "key": "project",
            "value": "doc-smith"
        },
        {
            "key": "tutorial",
            "value": "amazon-ecs-service-connect"
        }
    ]
}
EOF
```

```
    SERVICE_OUTPUT=$(aws ecs create-service --cli-input-json file:///tmp/service-
definition.json 2>&1)
    check_for_errors "$SERVICE_OUTPUT" "create-service"

    track_resource "SERVICE:service-connect-nginx-service"
    log "Created ECS service: service-connect-nginx-service"

    wait_for_resource "service" "service-connect-nginx-service"

    # Clean up temporary file
    rm -f /tmp/service-definition.json
}

# Function to verify deployment
verify_deployment() {
    log "Verifying deployment..."

    # Check service status
    SERVICE_STATUS=$(aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].status' --output text)
    log "Service status: $SERVICE_STATUS"

    # Check running tasks
    RUNNING_COUNT=$(aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].runningCount' --output text)
    log "Running tasks: $RUNNING_COUNT"

    # Get task ARN
    TASK_ARN=$(aws ecs list-tasks \
        --cluster "$CLUSTER_NAME" \
        --service-name "service-connect-nginx-service" \
        --query 'taskArns[0]' --output text)

    if [[ "$TASK_ARN" != "None" && -n "$TASK_ARN" ]]; then
        log "Task ARN: $TASK_ARN"

        # Try to get task IP address
        TASK_IP=$(aws ecs describe-tasks \
            --cluster "$CLUSTER_NAME" \
```

```
            --tasks "$TASK_ARN" \
            --query 'tasks[0].attachments[0].details[?
name==`privateIPv4Address`].value' \
            --output text 2>/dev/null || echo "")

        if [[ -n "$TASK_IP" && "$TASK_IP" != "None" ]]; then
            log "Task IP address: $TASK_IP"
        else
            log "Could not retrieve task IP address"
        fi
    fi

    # Check Service Connect namespace
    NAMESPACE_STATUS=$(aws servicediscovery list-namespaces \
        --filters Name=TYPE,Values=HTTP \
        --query "Namespaces[?Name=='$NAMESPACE_NAME'].Id" --output text 2>/dev/
null || echo "")

    if [[ -n "$NAMESPACE_STATUS" && "$NAMESPACE_STATUS" != "None" ]]; then
        log "Service Connect namespace '$NAMESPACE_NAME' is active"
    else
        log "Service Connect namespace '$NAMESPACE_NAME' not found or not active"
    fi

    # Display Service Connect configuration
    log "Service Connect configuration:"
    aws ecs describe-services \
        --cluster "$CLUSTER_NAME" \
        --services "service-connect-nginx-service" \
        --query 'services[0].serviceConnectConfiguration' 2>/dev/null || true
}

# Function to display created resources
display_resources() {
    echo ""
    echo "============================================="
    echo "CREATED RESOURCES"
    echo "============================================="
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "- $resource"
    done
    echo "============================================="
    echo ""
}
```

```
# Function to cleanup resources
cleanup_resources() {
    log "Starting cleanup process..."

    # Delete resources in reverse order of creation
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        resource="${CREATED_RESOURCES[i]}"
        resource_type=$(echo "$resource" | cut -d':' -f1)
        resource_id=$(echo "$resource" | cut -d':' -f2)

        log "Cleaning up $resource_type: $resource_id"

        case "$resource_type" in
            "SERVICE")
                aws ecs update-service --cluster "$CLUSTER_NAME" --service
 "$resource_id" --desired-count 0 2>&1 | grep -qi "error" && log "Warning: Failed
 to scale down service $resource_id"
                aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
 "$resource_id" 2>/dev/null || true
                aws ecs delete-service --cluster "$CLUSTER_NAME" --service
 "$resource_id" --force 2>&1 | grep -qi "error" && log "Warning: Failed to delete
 service $resource_id"
                ;;
            "TASK_DEF")
                TASK_DEF_ARNS=$(aws ecs list-task-definitions --family-prefix
 "$resource_id" --query 'taskDefinitionArns' --output text 2>/dev/null)
                for arn in $TASK_DEF_ARNS; do
                    aws ecs deregister-task-definition --task-definition "$arn"
 >/dev/null 2>&1 || true
                done
                ;;
            "ROLE")
                aws iam detach-role-policy --role-name "$resource_id" --policy-
arn "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy" 2>/
dev/null || true
                aws iam delete-role --role-name "$resource_id" 2>&1 | grep -qi
 "error" && log "Warning: Failed to delete role $resource_id"
                ;;
            "IAM_ROLE")
                aws iam detach-role-policy --role-name "$resource_id" --policy-
arn "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy" 2>/
dev/null || true
```

```
                aws iam delete-role --role-name "$resource_id" 2>&1 | grep -qi
 "error" && log "Warning: Failed to delete role $resource_id"
                ;;
            "CLUSTER")
                aws ecs delete-cluster --cluster "$resource_id" 2>&1 | grep -qi
 "error" && log "Warning: Failed to delete cluster $resource_id"
                ;;
            "SG")
                for attempt in 1 2 3 4 5; do
                    if aws ec2 delete-security-group --group-id "$resource_id"
 2>/dev/null; then
                        break
                    fi
                    log "Security group $resource_id still has dependencies,
 retrying in 30s ($attempt/5)..."
                    sleep 30
                done
                ;;
            "LOG_GROUP")
                aws logs delete-log-group --log-group-name "$resource_id" 2>&1 |
 grep -qi "error" && log "Warning: Failed to delete log group $resource_id"
                ;;
            "NAMESPACE")
                # First, delete any services in the namespace
                NAMESPACE_SERVICES=$(aws servicediscovery list-services \
                    --filters Name=NAMESPACE_ID,Values="$resource_id" \
                    --query 'Services[].Id' --output text 2>/dev/null || echo "")

                if [[ -n "$NAMESPACE_SERVICES" && "$NAMESPACE_SERVICES" !=
 "None" ]]; then
                    for service_id in $NAMESPACE_SERVICES; do
                        aws servicediscovery delete-service --id "$service_id" >/
dev/null 2>&1 || true
                        sleep 2
                    done
                fi

                # Then delete the namespace
                aws servicediscovery delete-namespace --id "$resource_id" >/dev/
null 2>&1 || true
                ;;
        esac

        sleep 2  # Brief pause between deletions
```

```
    done

    # Clean up temporary files
    rm -f /tmp/ecs-task-trust-policy.json
    rm -f /tmp/task-definition.json
    rm -f /tmp/service-definition.json

    log "Cleanup completed"
}

# Main execution
main() {
    log "Starting $SCRIPT_NAME v4 (Default VPC)"
    log "Region: $REGION"
    log "Log file: $LOG_FILE"

    # Get AWS account ID
    get_account_id

    # Setup infrastructure using default VPC
    setup_default_vpc_infrastructure

    # Create CloudWatch log groups
    create_log_groups

    # Create ECS cluster
    create_ecs_cluster

    # Create IAM roles
    create_iam_roles

    # Create task definition
    create_task_definition

    # Create ECS service
    create_ecs_service

    # Verify deployment
    verify_deployment

    log "Tutorial completed successfully!"

    # Display created resources
    display_resources
```

```
    # Ask user if they want to clean up
    echo ""
    echo "==========================================="
    echo "CLEANUP CONFIRMATION"
    echo "==========================================="
    echo "Do you want to clean up all created resources? (y/n): "
    CLEANUP_CHOICE="y"

    if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
        cleanup_resources
        log "All resources have been cleaned up"
    else
        log "Resources left intact. You can clean them up later by running the
 cleanup function."
        echo ""
        echo "To clean up resources later, you can use the AWS CLI commands or
 the AWS Management Console."
        echo "Remember to delete resources in the correct order to avoid
 dependency issues."
    fi
}

# Make script executable and run
chmod +x "$0"
main "$@"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - AuthorizeSecurityGroupIngress
  - CreateCluster
  - CreateLogGroup
  - CreateRole
  - CreateSecurityGroup
  - CreateService
  - DeleteCluster
  - DeleteLogGroup
  - DeleteNamespace

- DeleteRole
- DeleteSecurityGroup
- DeleteService (AWS Cloud Map)
- DeleteService (Amazon ECS)
- DeregisterTaskDefinition
- DescribeClusters
- DescribeServices
- DescribeSubnets
- DescribeTasks
- DescribeVpcs
- DetachRolePolicy
- GetCallerIdentity
- GetRole
- ListNamespaces
- ListServices
- ListTaskDefinitions
- ListTasks
- RegisterTaskDefinition
- UpdateService
- Wait (Amazon EC2)
- Wait (Amazon ECS)

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Construct a URL with AWS STS for federated users using an AWS SDK

The following code example shows how to:

- Create an IAM role that grants read-only access to the current account's Amazon S3 resources.
- Get a security token from the AWS federation endpoint.
- Construct a URL that can be used to access the console with federated credentials.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a role that grants read-only access to the current account's S3 resources.

```
def setup(iam_resource):
    """
    Creates a role that can be assumed by the current user.
    Attaches a policy that allows only Amazon S3 read-only access.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM)
 instance
                         that has the permission to create a role.
    :return: The newly created role.
    """
    role = iam_resource.create_role(
        RoleName=unique_name("role"),
        AssumeRolePolicyDocument=json.dumps(
            {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {"AWS": iam_resource.CurrentUser().arn},
                        "Action": "sts:AssumeRole",
                    }
                ],
            }
        ),
    )
    role.attach_policy(PolicyArn="arn:aws:iam::aws:policy/
AmazonS3ReadOnlyAccess")
    print(f"Created role {role.name}.")

    print("Give AWS time to propagate these new resources and connections.",
 end="")
```

```
    progress_bar(10)

    return role
```

Get a security token from the AWS federation endpoint and construct a URL that can be used to access the console with federated credentials.

```
def construct_federated_url(assume_role_arn, session_name, issuer, sts_client):
    """
    Constructs a URL that gives federated users direct access to the AWS
 Management
    Console.

    1. Acquires temporary credentials from AWS Security Token Service (AWS STS)
 that
       can be used to assume a role with limited permissions.
    2. Uses the temporary credentials to request a sign-in token from the
       AWS federation endpoint.
    3. Builds a URL that can be used in a browser to navigate to the AWS
 federation
       endpoint, includes the sign-in token for authentication, and redirects to
       the AWS Management Console with permissions defined by the role that was
       specified in step 1.

    :param assume_role_arn: The role that specifies the permissions that are
 granted.
                            The current user must have permission to assume the
 role.
    :param session_name: The name for the STS session.
    :param issuer: The organization that issues the URL.
    :param sts_client: A Boto3 STS instance that can assume the role.
    :return: The federated URL.
    """
    response = sts_client.assume_role(
        RoleArn=assume_role_arn, RoleSessionName=session_name
    )
    temp_credentials = response["Credentials"]
    print(f"Assumed role {assume_role_arn} and got temporary credentials.")

    session_data = {
```

```
        "sessionId": temp_credentials["AccessKeyId"],
        "sessionKey": temp_credentials["SecretAccessKey"],
        "sessionToken": temp_credentials["SessionToken"],
    }
    aws_federated_signin_endpoint = "https://signin.aws.amazon.com/federation"

    # Make a request to the AWS federation endpoint to get a sign-in token.
    # The requests.get function URL-encodes the parameters and builds the query
 string
    # before making the request.
    response = requests.get(
        aws_federated_signin_endpoint,
        params={
            "Action": "getSigninToken",
            "SessionDuration": str(datetime.timedelta(hours=12).seconds),
            "Session": json.dumps(session_data),
        },
    )
    signin_token = json.loads(response.text)
    print(f"Got a sign-in token from the AWS sign-in federation endpoint.")

    # Make a federated URL that can be used to sign into the AWS Management
 Console.
    query_string = urllib.parse.urlencode(
        {
            "Action": "login",
            "Issuer": issuer,
            "Destination": "https://console.aws.amazon.com/",
            "SigninToken": signin_token["SigninToken"],
        }
    )
    federated_url = f"{aws_federated_signin_endpoint}?{query_string}"
    return federated_url
```

Destroy the resources created for the demo.

```
def teardown(role):
    """
    Removes all resources created during setup.
```

```
    :param role: The demo role.
    """
    for attached in role.attached_policies.all():
        role.detach_policy(PolicyArn=attached.arn)
        print(f"Detached {attached.policy_name}.")
    role.delete()
    print(f"Deleted {role.name}.")
```

Run this scenario by using the previously defined functions.

```
def usage_demo():
    """Drives the demonstration."""
    print("-" * 88)
    print(f"Welcome to the AWS Security Token Service federated URL demo.")
    print("-" * 88)
    iam_resource = boto3.resource("iam")
    role = setup(iam_resource)
    sts_client = boto3.client("sts")
    try:
        federated_url = construct_federated_url(
            role.arn, "AssumeRoleDemoSession", "example.org", sts_client
        )
        print(
            "Constructed a federated URL that can be used to connect to the "
            "AWS Management Console with role-defined permissions:"
        )
        print("-" * 88)
        print(federated_url)
        print("-" * 88)
        _ = input(
            "Copy and paste the above URL into a browser to open the AWS "
            "Management Console with limited permissions. When done, press "
            "Enter to clean up and complete this demo."
        )
    finally:
        teardown(role)
        print("Thanks for watching!")
```

- For API details, see AssumeRole in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a container task for the serverless launch type

The following code example shows how to:

- Create the cluster
- Create a task definition
- Create the service
- Clean up

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon ECS Fargate Tutorial Script - Version 5
# This script creates an ECS cluster, task definition, and service using Fargate
 launch type
# Fixed version with proper resource dependency handling during cleanup
# Security improvements applied

set -e  # Exit on any error

# Initialize logging
LOG_FILE="ecs-fargate-tutorial-v5.log"
exec > >(tee -a "$LOG_FILE") 2>&1
```

```
echo "Starting Amazon ECS Fargate tutorial at $(date)"
echo "Log file: $LOG_FILE"

# Generate random identifier for unique resource names
RANDOM_ID=$(openssl rand -hex 6)
CLUSTER_NAME="fargate-cluster-${RANDOM_ID}"
SERVICE_NAME="fargate-service-${RANDOM_ID}"
TASK_FAMILY="sample-fargate-${RANDOM_ID}"
SECURITY_GROUP_NAME="ecs-fargate-sg-${RANDOM_ID}"

# Array to track created resources for cleanup
CREATED_RESOURCES=()

# Function to log and execute commands with input validation
execute_command() {
    local cmd="$1"
    local description="$2"

    # Validate that cmd is not empty
    if [[ -z "$cmd" ]]; then
        echo "ERROR: Command is empty"
        return 1
    fi

    echo ""
    echo "==========================================="
    echo "EXECUTING: $description"
    echo "COMMAND: $cmd"
    echo "==========================================="

    local output
    local exit_code
    set +e  # Temporarily disable exit on error
    output=$(eval "$cmd" 2>&1)
    exit_code=$?
    set -e  # Re-enable exit on error

    if [[ $exit_code -eq 0 ]]; then
        echo "SUCCESS: $description"
        echo "OUTPUT: $output"
        return 0
    else
        echo "FAILED: $description"
        echo "EXIT CODE: $exit_code"
```

```
        echo "OUTPUT: $output"
        return 1
    fi
}

# Function to check for actual AWS API errors in command output
check_for_aws_errors() {
    local output="$1"
    local description="$2"

    # Look for specific AWS error patterns, not just the word "error"
    if echo "$output" | grep -qi "An error occurred\|InvalidParameter\|
AccessDenied\|ResourceNotFound\|ValidationException"; then
        echo "AWS API ERROR detected in output for: $description"
        echo "Output: $output"
        return 1
    fi
    return 0
}

# Function to safely extract JSON values
safe_json_extract() {
    local json="$1"
    local key="$2"
    local value

    value=$(echo "$json" | grep -o "\"$key\": \"[^\"]*\"" | cut -d'"' -f4 2>/dev/
null || echo "")
    echo "$value"
}

# Function to wait for network interfaces to be cleaned up
wait_for_network_interfaces_cleanup() {
    local security_group_id="$1"
    local max_attempts=30
    local attempt=1

    # Validate security group ID format
    if [[ ! "$security_group_id" =~ ^sg-[a-z0-9]{8,17}$ ]]; then
        echo "ERROR: Invalid security group ID format: $security_group_id"
        return 1
    fi

    echo "Waiting for network interfaces to be cleaned up..."
```

```
    while [[ $attempt -le $max_attempts ]]; do
        echo "Attempt $attempt/$max_attempts: Checking for dependent network
 interfaces..."

        # Check if there are any network interfaces still using this security
 group
        local eni_count
        eni_count=$(aws ec2 describe-network-interfaces \
            --filters "Name=group-id,Values=$security_group_id" \
            --query "length(NetworkInterfaces)" \
            --output text 2>/dev/null || echo "0")

        if [[ "$eni_count" == "0" ]]; then
            echo "No network interfaces found using security group
 $security_group_id"
            return 0
        else
            echo "Found $eni_count network interface(s) still using security
 group $security_group_id"
            echo "Waiting 10 seconds before next check..."
            sleep 10
            ((attempt++))
        fi
    done

    echo "WARNING: Network interfaces may still be attached after $max_attempts
 attempts"
    echo "This is normal and the security group deletion will be retried"
    return 1
}

# Function to retry security group deletion with exponential backoff
retry_security_group_deletion() {
    local security_group_id="$1"
    local max_attempts=10
    local attempt=1
    local wait_time=5

    # Validate security group ID format
    if [[ ! "$security_group_id" =~ ^sg-[a-z0-9]{8,17}$ ]]; then
        echo "ERROR: Invalid security group ID format: $security_group_id"
        return 1
    fi
```

```
    while [[ $attempt -le $max_attempts ]]; do
        echo "Attempt $attempt/$max_attempts: Trying to delete security group
 $security_group_id"

        if execute_command "aws ec2 delete-security-group --group-id
 '$security_group_id'" "Delete security group (attempt $attempt)"; then
            echo "Successfully deleted security group $security_group_id"
            return 0
        else
            if [[ $attempt -eq $max_attempts ]]; then
                echo "FAILED: Could not delete security group $security_group_id
 after $max_attempts attempts"
                echo "This may be due to network interfaces that are still being
 cleaned up by AWS"
                echo "You can manually delete it later using: aws ec2 delete-
security-group --group-id $security_group_id"
                return 1
            else
                echo "Waiting $wait_time seconds before retry..."
                sleep $wait_time
                wait_time=$((wait_time * 2))  # Exponential backoff
                ((attempt++))
            fi
        fi
    done
}

# Function to cleanup resources with proper dependency handling
cleanup_resources() {
    echo ""
    echo "============================================"
    echo "CLEANUP PROCESS"
    echo "============================================"
    echo "The following resources were created:"
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "  - $resource"
    done
    echo ""
    echo "Auto-confirming cleanup of all created resources..."

    CLEANUP_CHOICE="y"

    if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
```

```
        echo "Starting cleanup process..."

        # Step 1: Scale service to 0 tasks first, then delete service
        if [[ " ${CREATED_RESOURCES[*]} " =~ " ECS Service: $SERVICE_NAME " ]];
then
            echo ""
            echo "Step 1: Scaling service to 0 tasks..."
            if execute_command "aws ecs update-service --cluster '$CLUSTER_NAME'
--service '$SERVICE_NAME' --desired-count 0" "Scale service to 0 tasks"; then
                echo "Waiting for service to stabilize after scaling to 0..."
                execute_command "aws ecs wait services-stable --cluster
'$CLUSTER_NAME' --services '$SERVICE_NAME'" "Wait for service to stabilize"

                echo "Deleting service..."
                execute_command "aws ecs delete-service --cluster '$CLUSTER_NAME'
--service '$SERVICE_NAME'" "Delete ECS service"
            else
                echo "WARNING: Failed to scale service. Attempting to delete
anyway..."
                execute_command "aws ecs delete-service --cluster '$CLUSTER_NAME'
--service '$SERVICE_NAME' --force" "Force delete ECS service"
            fi
        fi

        # Step 2: Wait a bit for tasks to fully terminate
        echo ""
        echo "Step 2: Waiting for tasks to fully terminate..."
        sleep 15

        # Step 3: Delete cluster
        if [[ " ${CREATED_RESOURCES[*]} " =~ " ECS Cluster: $CLUSTER_NAME " ]];
then
            echo ""
            echo "Step 3: Deleting cluster..."
            execute_command "aws ecs delete-cluster --cluster '$CLUSTER_NAME'"
"Delete ECS cluster"
        fi

        # Step 4: Wait for network interfaces to be cleaned up, then delete
security group
        if [[ -n "$SECURITY_GROUP_ID" && "$SECURITY_GROUP_ID" != "None" ]]; then
            echo ""
            echo "Step 4: Cleaning up security group..."
```

```
            # First, wait for network interfaces to be cleaned up
            wait_for_network_interfaces_cleanup "$SECURITY_GROUP_ID"

            # Then retry security group deletion with backoff
            retry_security_group_deletion "$SECURITY_GROUP_ID"
        fi

        # Step 5: Clean up task definition (deregister all revisions)
        if [[ " ${CREATED_RESOURCES[*]} " =~ " Task Definition: $TASK_FAMILY
 " ]]; then
            echo ""
            echo "Step 5: Deregistering task definition revisions..."

            # Get all revisions of the task definition
            local revisions
            revisions=$(aws ecs list-task-definitions --family-prefix
 "$TASK_FAMILY" --query "taskDefinitionArns" --output text 2>/dev/null || echo
 "")

            if [[ -n "$revisions" && "$revisions" != "None" ]]; then
                for revision_arn in $revisions; do
                    echo "Deregistering task definition: $revision_arn"
                    execute_command "aws ecs deregister-task-definition --task-
definition '$revision_arn'" "Deregister task definition $revision_arn" || true
                done
            else
                echo "No task definition revisions found to deregister"
            fi
        fi

        echo ""
        echo "============================================"
        echo "CLEANUP COMPLETED"
        echo "============================================"
        echo "All resources have been cleaned up successfully!"

    else
        echo "Cleanup skipped. Resources remain active."
        echo ""
        echo "To clean up manually later, use the following commands in order:"
        echo "1. Scale service to 0: aws ecs update-service --cluster
 $CLUSTER_NAME --service $SERVICE_NAME --desired-count 0"
        echo "2. Wait for stability: aws ecs wait services-stable --cluster
 $CLUSTER_NAME --services $SERVICE_NAME"
```

```
        echo "3. Delete service: aws ecs delete-service --cluster $CLUSTER_NAME
 --service $SERVICE_NAME"
        echo "4. Delete cluster: aws ecs delete-cluster --cluster $CLUSTER_NAME"
        echo "5. Wait 2-3 minutes, then delete security group: aws ec2 delete-
security-group --group-id $SECURITY_GROUP_ID"
        if [[ " ${CREATED_RESOURCES[*]} " =~ " Task Definition: $TASK_FAMILY
 " ]]; then
            echo "6. Deregister task definitions: aws ecs list-task-definitions
 --family-prefix $TASK_FAMILY"
            echo "   Then for each ARN: aws ecs deregister-task-definition --
task-definition <ARN>"
        fi
    fi
}

# Trap to handle script interruption
trap cleanup_resources EXIT

echo "Using random identifier: $RANDOM_ID"
echo "Cluster name: $CLUSTER_NAME"
echo "Service name: $SERVICE_NAME"
echo "Task family: $TASK_FAMILY"

# Step 1: Ensure ECS task execution role exists
echo ""
echo "==========================================="
echo "STEP 1: VERIFY ECS TASK EXECUTION ROLE"
echo "==========================================="

ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
if [[ ! "$ACCOUNT_ID" =~ ^[0-9]{12}$ ]]; then
    echo "ERROR: Invalid AWS Account ID retrieved: $ACCOUNT_ID"
    exit 1
fi

EXECUTION_ROLE_ARN="arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskExecutionRole"

# Check if role exists
if aws iam get-role --role-name ecsTaskExecutionRole >/dev/null 2>&1; then
    echo "ECS task execution role already exists"
else
    echo "Creating ECS task execution role..."

    # Create trust policy with strict validation
```

```
    cat > trust-policy.json << 'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF

    # Validate JSON before using
    if ! jq empty trust-policy.json 2>/dev/null; then
        echo "ERROR: Invalid JSON in trust policy"
        rm -f trust-policy.json
        exit 1
    fi

    execute_command "aws iam create-role --role-name ecsTaskExecutionRole --
assume-role-policy-document file://trust-policy.json" "Create ECS task execution
 role"

    aws iam tag-role --role-name ecsTaskExecutionRole --tags
 Key=project,Value=doc-smith Key=tutorial,Value=amazon-ecs-fargate-linux

    execute_command "aws iam attach-role-policy --role-name
 ecsTaskExecutionRole --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy" "Attach ECS task execution policy"

    # Clean up temporary file securely
    shred -vfz -n 3 trust-policy.json 2>/dev/null || rm -f trust-policy.json

    CREATED_RESOURCES+=("IAM Role: ecsTaskExecutionRole")
fi

# Step 2: Create ECS cluster
echo ""
echo "==========================================="
echo "STEP 2: CREATE ECS CLUSTER"
echo "==========================================="
```

```
CLUSTER_OUTPUT=$(execute_command "aws ecs create-cluster --cluster-name
 '$CLUSTER_NAME' --tags key=project,value=doc-smith key=tutorial,value=amazon-
ecs-fargate-linux" "Create ECS cluster")
check_for_aws_errors "$CLUSTER_OUTPUT" "Create ECS cluster"

CREATED_RESOURCES+=("ECS Cluster: $CLUSTER_NAME")

# Step 3: Create task definition
echo ""
echo "=========================================="
echo "STEP 3: CREATE TASK DEFINITION"
echo "=========================================="

# Create task definition JSON with validated inputs
cat > task-definition.json << EOF
{
    "family": "$TASK_FAMILY",
    "networkMode": "awsvpc",
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "256",
    "memory": "512",
    "executionRoleArn": "$EXECUTION_ROLE_ARN",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "public.ecr.aws/docker/library/httpd:2.4-alpine",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": ["sh", "-c"],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container
 in Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html
 && httpd-foreground\""
            ],
```

```
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/fargate-sample",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ]
}
EOF

# Validate JSON before using
if ! jq empty task-definition.json 2>/dev/null; then
    echo "ERROR: Invalid JSON in task definition"
    rm -f task-definition.json
    exit 1
fi

TASK_DEF_OUTPUT=$(execute_command "aws ecs register-task-definition --cli-input-
json file://task-definition.json" "Register task definition")
check_for_aws_errors "$TASK_DEF_OUTPUT" "Register task definition"

# Clean up temporary file securely
shred -vfz -n 3 task-definition.json 2>/dev/null || rm -f task-definition.json

CREATED_RESOURCES+=("Task Definition: $TASK_FAMILY")

# Step 4: Set up networking
echo ""
echo "============================================="
echo "STEP 4: SET UP NETWORKING"
echo "============================================="

# Get default VPC ID
VPC_ID=$(aws ec2 describe-vpcs --filters "Name=is-default,Values=true" --query
 "Vpcs[0].VpcId" --output text)
if [[ "$VPC_ID" == "None" || -z "$VPC_ID" ]]; then
    echo "ERROR: No default VPC found. Please create a default VPC or specify a
 custom VPC."
    exit 1
fi
```

```
# Validate VPC ID format
if [[ ! "$VPC_ID" =~ ^vpc-[a-z0-9]{8,17}$ ]]; then
    echo "ERROR: Invalid VPC ID format: $VPC_ID"
    exit 1
fi

echo "Using default VPC: $VPC_ID"

# Create security group with restricted access
# Note: This allows HTTP access from anywhere for demo purposes
# In production, restrict source to specific IP ranges or security groups
SECURITY_GROUP_OUTPUT=$(execute_command "aws ec2 create-security-group --
group-name '$SECURITY_GROUP_NAME' --description 'Security group for ECS
 Fargate tutorial - HTTP access' --vpc-id '$VPC_ID' --tag-specifications
 'ResourceType=security-group,Tags=[{Key=project,Value=doc-smith},
{Key=tutorial,Value=amazon-ecs-fargate-linux}]'" "Create security group")
check_for_aws_errors "$SECURITY_GROUP_OUTPUT" "Create security group"

SECURITY_GROUP_ID=$(echo "$SECURITY_GROUP_OUTPUT" | grep -o '"GroupId": "[^"]*"'
 | head -1 | cut -d'"' -f4)
if [[ -z "$SECURITY_GROUP_ID" || "$SECURITY_GROUP_ID" == "None" ]]; then
    SECURITY_GROUP_ID=$(aws ec2 describe-security-groups --group-names
 "$SECURITY_GROUP_NAME" --filters "Name=vpc-id,Values=$VPC_ID" --query
 "SecurityGroups[0].GroupId" --output text)
fi

# Validate security group ID format
if [[ ! "$SECURITY_GROUP_ID" =~ ^sg-[a-z0-9]{8,17}$ ]]; then
    echo "ERROR: Invalid security group ID format: $SECURITY_GROUP_ID"
    exit 1
fi

echo "Created security group: $SECURITY_GROUP_ID"
CREATED_RESOURCES+=("Security Group: $SECURITY_GROUP_ID")

# Add HTTP inbound rule
# WARNING: This allows HTTP access from anywhere (0.0.0.0/0)
# In production environments, restrict this to specific IP ranges
execute_command "aws ec2 authorize-security-group-ingress --group-id
 '$SECURITY_GROUP_ID' --protocol tcp --port 80 --cidr 0.0.0.0/0" "Add HTTP
 inbound rule to security group"

# Get subnet IDs from default VPC
echo "Getting subnet IDs from default VPC..."
```

```
SUBNET_IDS_RAW=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=$VPC_ID"
 --query "Subnets[*].SubnetId" --output text)
if [[ -z "$SUBNET_IDS_RAW" ]]; then
    echo "ERROR: No subnets found in default VPC"
    exit 1
fi

# Convert to proper comma-separated format, handling both spaces and tabs
SUBNET_IDS_COMMA=$(echo "$SUBNET_IDS_RAW" | tr -s '[:space:]' ',' | sed 's/,$//')
echo "Raw subnet IDs: $SUBNET_IDS_RAW"
echo "Formatted subnet IDs: $SUBNET_IDS_COMMA"

# Validate subnet IDs format
if [[ ! "$SUBNET_IDS_COMMA" =~ ^subnet-[a-z0-9]+(,subnet-[a-z0-9]+)*$ ]]; then
    echo "ERROR: Invalid subnet ID format: $SUBNET_IDS_COMMA"
    exit 1
fi

# Step 5: Create ECS service
echo ""
echo "========================================="
echo "STEP 5: CREATE ECS SERVICE"
echo "========================================="

# Create the service with proper JSON formatting for network configuration
SERVICE_CMD="aws ecs create-service --cluster '$CLUSTER_NAME' --service-name
 '$SERVICE_NAME' --task-definition '$TASK_FAMILY' --desired-count 1 --launch-
type FARGATE --network-configuration '{\"awsvpcConfiguration\":{\"subnets
\":[\"$(echo "$SUBNET_IDS_COMMA" | sed 's/,/","/g')\"],\"securityGroups
\":[\"$SECURITY_GROUP_ID\"],\"assignPublicIp\":\"ENABLED\"}}' --tags
 key=project,value=doc-smith key=tutorial,value=amazon-ecs-fargate-linux"

echo "Service creation command: $SERVICE_CMD"

SERVICE_OUTPUT=$(execute_command "$SERVICE_CMD" "Create ECS service")
check_for_aws_errors "$SERVICE_OUTPUT" "Create ECS service"

CREATED_RESOURCES+=("ECS Service: $SERVICE_NAME")

# Step 6: Wait for service to stabilize and get public IP
echo ""
echo "========================================="
echo "STEP 6: WAIT FOR SERVICE AND GET PUBLIC IP"
echo "========================================="
```

```
echo "Waiting for service to stabilize (this may take a few minutes)..."
execute_command "aws ecs wait services-stable --cluster '$CLUSTER_NAME' --
services '$SERVICE_NAME'" "Wait for service to stabilize"

# Get task ARN
TASK_ARN=$(aws ecs list-tasks --cluster "$CLUSTER_NAME" --service-name
 "$SERVICE_NAME" --query "taskArns[0]" --output text)
if [[ "$TASK_ARN" == "None" || -z "$TASK_ARN" ]]; then
    echo "ERROR: No running tasks found for service"
    exit 1
fi

echo "Task ARN: $TASK_ARN"

# Get network interface ID
ENI_ID=$(aws ecs describe-tasks --cluster "$CLUSTER_NAME" --tasks "$TASK_ARN"
 --query "tasks[0].attachments[0].details[?name=='networkInterfaceId'].value" --
output text)
if [[ "$ENI_ID" == "None" || -z "$ENI_ID" ]]; then
    echo "ERROR: Could not retrieve network interface ID"
    exit 1
fi

# Validate ENI ID format
if [[ ! "$ENI_ID" =~ ^eni-[a-z0-9]{8,17}$ ]]; then
    echo "ERROR: Invalid network interface ID format: $ENI_ID"
    exit 1
fi

echo "Network Interface ID: $ENI_ID"

# Get public IP
PUBLIC_IP=$(aws ec2 describe-network-interfaces --network-interface-ids "$ENI_ID"
 --query "NetworkInterfaces[0].Association.PublicIp" --output text)
if [[ "$PUBLIC_IP" == "None" || -z "$PUBLIC_IP" ]]; then
    echo "WARNING: No public IP assigned to the task"
    echo "The task may be in a private subnet or public IP assignment failed"
else
    # Validate IP format
    if [[ ! "$PUBLIC_IP" =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]];
 then
        echo "ERROR: Invalid IP address format: $PUBLIC_IP"
        PUBLIC_IP=""
```

```
    else
        echo ""
        echo "========================================="
        echo "SUCCESS! APPLICATION IS RUNNING"
        echo "========================================="
        echo "Your application is available at: http://$PUBLIC_IP"
        echo "You can test it by opening this URL in your browser"
        echo ""
    fi
fi

# Display service information
echo ""
echo "========================================="
echo "SERVICE INFORMATION"
echo "========================================="
execute_command "aws ecs describe-services --cluster '$CLUSTER_NAME' --services
 '$SERVICE_NAME'" "Get service details"

echo ""
echo "========================================="
echo "TUTORIAL COMPLETED SUCCESSFULLY"
echo "========================================="
echo "Resources created:"
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "  - $resource"
done

if [[ -n "$PUBLIC_IP" && "$PUBLIC_IP" != "None" ]]; then
    echo ""
    echo "Application URL: http://$PUBLIC_IP"
fi

echo ""
echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - AuthorizeSecurityGroupIngress
  - CreateCluster
  - CreateRole

- CreateSecurityGroup
- CreateService
- DeleteCluster
- DeleteSecurityGroup
- DeleteService
- DeregisterTaskDefinition
- DescribeNetworkInterfaces
- DescribeSecurityGroups
- DescribeServices
- DescribeSubnets
- DescribeTasks
- DescribeVpcs
- GetCallerIdentity
- GetRole
- ListTaskDefinitions
- ListTasks
- RegisterTaskDefinition
- UpdateService
- Wait

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a rest API with function proxy integration

The following code example shows how to:

- Create an IAM role for Lambda execution
- Create and deploy a Lambda function
- Create a REST API
- Configure Lambda proxy integration
- Deploy and test the API

- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

set -euo pipefail

# Simple API Gateway Lambda Integration Script
# This script creates a REST API with Lambda proxy integration

# Generate random identifiers
FUNCTION_NAME="GetStartedLambdaProxyIntegration-$(openssl rand -hex 4)"
ROLE_NAME="GetStartedLambdaBasicExecutionRole-$(openssl rand -hex 4)"
API_NAME="LambdaProxyAPI-$(openssl rand -hex 4)"

# Get AWS account info
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REGION=$(aws configure get region || echo "us-east-1")

# Validate inputs
if [[ -z "$ACCOUNT_ID" ]] || [[ -z "$REGION" ]]; then
    echo "Error: Failed to retrieve AWS account information" >&2
    exit 1
fi

echo "Creating Lambda function code..."

# Create Lambda function code with input validation
cat > lambda_function.py << 'EOF'
import json
import logging

logger = logging.getLogger()
```

```
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        logger.info("Received event: %s", json.dumps(event))

        greeter = 'World'

        # Safely retrieve greeter from query string parameters
        query_params = event.get('queryStringParameters') or {}
        if isinstance(query_params, dict) and 'greeter' in query_params:
            greeter_value = query_params.get('greeter')
            if isinstance(greeter_value, str) and greeter_value:
                greeter = greeter_value

        # Safely retrieve greeter from multi-value headers
        multi_headers = event.get('multiValueHeaders') or {}
        if isinstance(multi_headers, dict) and 'greeter' in multi_headers:
            greeter_list = multi_headers.get('greeter', [])
            if isinstance(greeter_list, list) and greeter_list:
                greeter = " and ".join(str(g) for g in greeter_list if g)

        # Safely retrieve greeter from headers
        headers = event.get('headers') or {}
        if isinstance(headers, dict) and 'greeter' in headers:
            greeter_value = headers.get('greeter')
            if isinstance(greeter_value, str) and greeter_value:
                greeter = greeter_value

        # Safely retrieve greeter from body
        body = event.get('body')
        if body and isinstance(body, str):
            try:
                body_dict = json.loads(body)
                if isinstance(body_dict, dict) and 'greeter' in body_dict:
                    greeter_value = body_dict.get('greeter')
                    if isinstance(greeter_value, str) and greeter_value:
                        greeter = greeter_value
            except (json.JSONDecodeError, ValueError) as e:
                logger.warning("Failed to parse body: %s", str(e))

        # Sanitize greeter to prevent injection
        greeter = greeter.replace('"', '\\"').replace("'", "\\'")
```

```
        response = {
            "statusCode": 200,
            "headers": {
                "Content-Type": "application/json"
            },
            "body": json.dumps({"message": f"Hello, {greeter}!"})
        }

        logger.info("Response: %s", json.dumps(response))
        return response

    except Exception as e:
        logger.error("Unexpected error: %s", str(e), exc_info=True)
        return {
            "statusCode": 500,
            "headers": {
                "Content-Type": "application/json"
            },
            "body": json.dumps({"error": "Internal server error"})
        }
EOF

# Create deployment package
zip -q function.zip lambda_function.py || {
    echo "Error: Failed to create function.zip" >&2
    exit 1
}

echo "Creating IAM role..."

# Create IAM trust policy
cat > trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
EOF

# Create IAM role with error handling
aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document file://trust-policy.json \
    --description "Temporary role for Lambda execution" || {
    echo "Error: Failed to create IAM role" >&2
    exit 1
}

aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=apigateway-lambda-integration

# Attach execution policy
aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || {
    echo "Error: Failed to attach IAM policy" >&2
    exit 1
}

# Wait for role propagation
sleep 15

echo "Creating Lambda function..."

# Create Lambda function with Python 3.11 (more recent runtime)
aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime python3.11 \
    --role "arn:aws:iam::$ACCOUNT_ID:role/$ROLE_NAME" \
    --handler lambda_function.lambda_handler \
    --zip-file fileb://function.zip \
    --timeout 30 \
    --memory-size 128 \
    --environment "Variables={LOG_LEVEL=INFO}" \
    --tags project=doc-smith,tutorial=apigateway-lambda-integration || {
    echo "Error: Failed to create Lambda function" >&2
    exit 1
}

echo "Creating API Gateway..."
```

```
# Create REST API with minimum logging
API_RESPONSE=$(aws apigateway create-rest-api \
    --name "$API_NAME" \
    --endpoint-configuration types=REGIONAL \
    --description "API for Lambda proxy integration tutorial" \
    --tags project=doc-smith,tutorial=apigateway-lambda-integration \
    --output json)

API_ID=$(echo "$API_RESPONSE" | grep -o '"id": "[^"]*"' | head -1 | cut -d'"' -
f4)

if [[ -z "$API_ID" ]]; then
    echo "Error: Failed to create API Gateway" >&2
    exit 1
fi

# Get root resource ID
ROOT_RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
 'items[?path==`/`].id' --output text)

# Create helloworld resource
aws apigateway create-resource \
    --rest-api-id "$API_ID" \
    --parent-id "$ROOT_RESOURCE_ID" \
    --path-part helloworld || {
    echo "Error: Failed to create resource" >&2
    exit 1
}

# Get resource ID
RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
 "items[?pathPart=='helloworld'].id" --output text)

# Create ANY method with no authorization (intentional for tutorial)
aws apigateway put-method \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
    --http-method ANY \
    --authorization-type NONE || {
    echo "Error: Failed to create method" >&2
    exit 1
}
```

```
# Set up Lambda proxy integration
LAMBDA_URI="arn:aws:apigateway:$REGION:lambda:path/2015-03-31/functions/
arn:aws:lambda:$REGION:$ACCOUNT_ID:function:$FUNCTION_NAME/invocations"

aws apigateway put-integration \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
    --http-method ANY \
    --type AWS_PROXY \
    --integration-http-method POST \
    --uri "$LAMBDA_URI" || {
    echo "Error: Failed to create integration" >&2
    exit 1
}

# Grant API Gateway permission to invoke Lambda
STATEMENT_ID="apigateway-invoke-$(openssl rand -hex 4)"
SOURCE_ARN="arn:aws:execute-api:$REGION:$ACCOUNT_ID:$API_ID/*/*"

aws lambda add-permission \
    --function-name "$FUNCTION_NAME" \
    --statement-id "$STATEMENT_ID" \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "$SOURCE_ARN" || {
    echo "Error: Failed to add Lambda permission" >&2
    exit 1
}

# Deploy API
aws apigateway create-deployment \
    --rest-api-id "$API_ID" \
    --stage-name test \
    --description "Test deployment" || {
    echo "Error: Failed to deploy API" >&2
    exit 1
}

echo "Testing API..."

# Test the API
INVOKE_URL="https://$API_ID.execute-api.$REGION.amazonaws.com/test/helloworld"

echo "API URL: $INVOKE_URL"
```

```
# Test with query parameter (with proper URL encoding)
echo "Testing with query parameter:"
curl -s -X GET "$INVOKE_URL?greeter=John" | jq . 2>/dev/null || curl -s -X GET
 "$INVOKE_URL?greeter=John"
echo ""

# Test with header
echo "Testing with header:"
curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -H 'greeter: John' | jq . 2>/dev/null || curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -H 'greeter: John'
echo ""

# Test with body
echo "Testing with POST body:"
curl -s -X POST "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}' | jq . 2>/dev/null || curl -s -X POST "$INVOKE_URL"
 \
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}'
echo ""

echo "Tutorial completed! API is available at: $INVOKE_URL"

# Cleanup
echo "Cleaning up resources..."

# Delete API
aws apigateway delete-rest-api --rest-api-id "$API_ID" || echo "Warning: Failed
 to delete API" >&2

# Delete Lambda function
aws lambda delete-function --function-name "$FUNCTION_NAME" || echo "Warning:
 Failed to delete Lambda function" >&2

# Detach policy and delete role
aws iam detach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || echo "Warning: Failed to detach policy" >&2
```

```
aws iam delete-role --role-name "$ROLE_NAME" || echo "Warning: Failed to delete
 role" >&2

# Clean up local files securely
rm -f lambda_function.py function.zip trust-policy.json

echo "Cleanup completed!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddPermission
  - AttachRolePolicy
  - CreateDeployment
  - CreateFunction
  - CreateResource
  - CreateRestApi
  - CreateRole
  - DeleteFunction
  - DeleteRestApi
  - DeleteRole
  - DetachRolePolicy
  - GetCallerIdentity
  - GetResources
  - GetRestApis
  - PutIntegration
  - PutMethod

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Creating a container service for virtual machine instances

The following code example shows how to:

- Create an ECS cluster
- Create and monitor a service
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# ECS EC2 Launch Type Tutorial Script - UPDATED VERSION
# This script demonstrates creating an ECS cluster, launching a container
 instance,
# registering a task definition, and creating a service using the EC2 launch
 type.
# Updated to match the tutorial draft with nginx web server and service creation.
#
# - UPDATED: Changed from sleep task to nginx web server with service

set -e  # Exit on any error

# Configuration
SCRIPT_NAME="ecs-ec2-tutorial"
LOG_FILE="${SCRIPT_NAME}-$(date +%Y%m%d-%H%M%S).log"
CLUSTER_NAME="tutorial-cluster-$(openssl rand -hex 4)"
TASK_FAMILY="nginx-task-$(openssl rand -hex 4)"
SERVICE_NAME="nginx-service-$(openssl rand -hex 4)"
KEY_PAIR_NAME="ecs-tutorial-key-$(openssl rand -hex 4)"
SECURITY_GROUP_NAME="ecs-tutorial-sg-$(openssl rand -hex 4)"

# Tags
PROJECT_TAG="doc-smith"
TUTORIAL_TAG="ecs-ec2"

# Get current AWS region dynamically
```

```
AWS_REGION=$(aws configure get region || echo "us-east-1")

# Resource tracking arrays
CREATED_RESOURCES=()
CLEANUP_ORDER=()

# Logging function
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

# Error handling function
handle_error() {
    local exit_code=$?
    log "ERROR: Script failed with exit code $exit_code"
    log "ERROR: Last command: $BASH_COMMAND"

    echo ""
    echo "========================================="
    echo "ERROR OCCURRED - ATTEMPTING CLEANUP"
    echo "========================================="
    echo "Resources created before error:"
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "  - $resource"
    done

    cleanup_resources
    exit $exit_code
}

# Set error trap
trap handle_error ERR

# FIXED: Enhanced cleanup function with proper error handling and logging
cleanup_resources() {
    log "Starting cleanup process..."
    local cleanup_errors=0

    # Delete service first (this will stop tasks automatically)
    if [[ -n "${SERVICE_ARN:-}" ]]; then
        log "Updating service to desired count 0: $SERVICE_NAME"
        if ! aws ecs update-service --cluster "$CLUSTER_NAME" --service
 "$SERVICE_NAME" --desired-count 0 2>>"$LOG_FILE"; then
            log "WARNING: Failed to update service desired count to 0"
```

```
            ((cleanup_errors++))
        else
            log "Waiting for service tasks to stop..."
            sleep 30  # Give time for tasks to stop
        fi

        log "Deleting service: $SERVICE_NAME"
        if ! aws ecs delete-service --cluster "$CLUSTER_NAME" --service
 "$SERVICE_NAME" 2>>"$LOG_FILE"; then
            log "WARNING: Failed to delete service $SERVICE_NAME"
            ((cleanup_errors++))
        fi
    fi

    # Stop and delete any remaining tasks
    if [[ -n "${TASK_ARN:-}" ]]; then
        log "Stopping task: $TASK_ARN"
        if ! aws ecs stop-task --cluster "$CLUSTER_NAME" --task "$TASK_ARN" --
reason "Tutorial cleanup" 2>>"$LOG_FILE"; then
            log "WARNING: Failed to stop task $TASK_ARN"
            ((cleanup_errors++))
        else
            log "Waiting for task to stop..."
            if ! aws ecs wait tasks-stopped --cluster "$CLUSTER_NAME" --tasks
 "$TASK_ARN" 2>>"$LOG_FILE"; then
                log "WARNING: Task stop wait failed for $TASK_ARN"
                ((cleanup_errors++))
            fi
        fi
    fi

    # Deregister task definition
    if [[ -n "${TASK_DEFINITION_ARN:-}" ]]; then
        log "Deregistering task definition: $TASK_DEFINITION_ARN"
        if ! aws ecs deregister-task-definition --task-definition
 "$TASK_DEFINITION_ARN" 2>>"$LOG_FILE"; then
            log "WARNING: Failed to deregister task definition
 $TASK_DEFINITION_ARN"
            ((cleanup_errors++))
        fi
    fi

    # Terminate EC2 instance
    if [[ -n "${INSTANCE_ID:-}" ]]; then
```

```
        log "Terminating EC2 instance: $INSTANCE_ID"
        if ! aws ec2 terminate-instances --instance-ids "$INSTANCE_ID"
2>>"$LOG_FILE"; then
            log "WARNING: Failed to terminate instance $INSTANCE_ID"
            ((cleanup_errors++))
        else
            log "Waiting for instance to terminate..."
            if ! aws ec2 wait instance-terminated --instance-ids "$INSTANCE_ID"
2>>"$LOG_FILE"; then
                log "WARNING: Instance termination wait failed for $INSTANCE_ID"
                ((cleanup_errors++))
            fi
        fi
    fi

    # Delete security group with retry logic
    if [[ -n "${SECURITY_GROUP_ID:-}" ]]; then
        log "Deleting security group: $SECURITY_GROUP_ID"
        local retry_count=0
        local max_retries=3

        while [[ $retry_count -lt $max_retries ]]; do
            if aws ec2 delete-security-group --group-id "$SECURITY_GROUP_ID"
2>>"$LOG_FILE"; then
                log "Successfully deleted security group"
                break
            else
                ((retry_count++))
                if [[ $retry_count -lt $max_retries ]]; then
                    log "Retry $retry_count/$max_retries: Waiting 10 seconds
before retrying security group deletion..."
                    sleep 10
                else
                    log "ERROR: Failed to delete security group after
$max_retries attempts"
                    ((cleanup_errors++))
                fi
            fi
        done
    fi

    # Delete key pair
    if [[ -n "${KEY_PAIR_NAME:-}" ]]; then
        log "Deleting key pair: $KEY_PAIR_NAME"
```

```
        if ! aws ec2 delete-key-pair --key-name "$KEY_PAIR_NAME" 2>>"$LOG_FILE";
 then
            log "WARNING: Failed to delete key pair $KEY_PAIR_NAME"
            ((cleanup_errors++))
        fi
        rm -f "${KEY_PAIR_NAME}.pem" 2>>"$LOG_FILE" || log "WARNING: Failed to
 remove local key file"
    fi

    # Delete ECS cluster
    if [[ -n "${CLUSTER_NAME:-}" ]]; then
        log "Deleting ECS cluster: $CLUSTER_NAME"
        if ! aws ecs delete-cluster --cluster "$CLUSTER_NAME" 2>>"$LOG_FILE";
 then
            log "WARNING: Failed to delete cluster $CLUSTER_NAME"
            ((cleanup_errors++))
        fi
    fi

    if [[ $cleanup_errors -eq 0 ]]; then
        log "Cleanup completed successfully"
    else
        log "Cleanup completed with $cleanup_errors warnings/errors. Check log
 file for details."
    fi
}

# Function to check prerequisites
check_prerequisites() {
    log "Checking prerequisites..."

    # Check AWS CLI
    if ! command -v aws &> /dev/null; then
        log "ERROR: AWS CLI is not installed"
        exit 1
    fi

    # Check AWS credentials
    if ! aws sts get-caller-identity &> /dev/null; then
        log "ERROR: AWS credentials not configured"
        exit 1
    fi

    # Get caller identity
```

```
    CALLER_IDENTITY=$(aws sts get-caller-identity --output text --query
 'Account')
    log "AWS Account: $CALLER_IDENTITY"
    log "AWS Region: $AWS_REGION"

    # Check for default VPC
    DEFAULT_VPC=$(aws ec2 describe-vpcs --filters "Name=is-default,Values=true"
 --query 'Vpcs[0].VpcId' --output text)
    if [[ "$DEFAULT_VPC" == "None" ]]; then
        log "ERROR: No default VPC found. Please create a VPC first."
        exit 1
    fi
    log "Using default VPC: $DEFAULT_VPC"

    # Get default subnet
    DEFAULT_SUBNET=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=
$DEFAULT_VPC" "Name=default-for-az,Values=true" --query 'Subnets[0].SubnetId' --
output text)
    if [[ "$DEFAULT_SUBNET" == "None" ]]; then
        log "ERROR: No default subnet found"
        exit 1
    fi
    log "Using default subnet: $DEFAULT_SUBNET"

    log "Prerequisites check completed successfully"
}

# Function to create ECS cluster
create_cluster() {
    log "Creating ECS cluster: $CLUSTER_NAME"

    CLUSTER_ARN=$(aws ecs create-cluster --cluster-name "$CLUSTER_NAME" --
tags key=project,value=$PROJECT_TAG key=tutorial,value=$TUTORIAL_TAG --query
 'cluster.clusterArn' --output text)

    if [[ -z "$CLUSTER_ARN" ]]; then
        log "ERROR: Failed to create cluster"
        exit 1
    fi

    log "Created cluster: $CLUSTER_ARN"
    CREATED_RESOURCES+=("ECS Cluster: $CLUSTER_NAME")
}
```

```
# Function to create key pair
create_key_pair() {
    log "Creating EC2 key pair: $KEY_PAIR_NAME"

    # FIXED: Set secure umask before key creation
    umask 077
    aws ec2 create-key-pair --key-name "$KEY_PAIR_NAME" --query 'KeyMaterial' --
output text > "${KEY_PAIR_NAME}.pem"
    chmod 400 "${KEY_PAIR_NAME}.pem"
    umask 022  # Reset umask

    log "Created key pair: $KEY_PAIR_NAME"
    CREATED_RESOURCES+=("EC2 Key Pair: $KEY_PAIR_NAME")

    aws ec2 create-tags --resources "$KEY_PAIR_NAME" --tags Key=project,Value=
$PROJECT_TAG Key=tutorial,Value=$TUTORIAL_TAG 2>>"$LOG_FILE" || log "WARNING:
 Failed to tag key pair"
}

# Function to create security group
create_security_group() {
    log "Creating security group: $SECURITY_GROUP_NAME"

    SECURITY_GROUP_ID=$(aws ec2 create-security-group \
        --group-name "$SECURITY_GROUP_NAME" \
        --description "ECS tutorial security group" \
        --vpc-id "$DEFAULT_VPC" \
        --tag-specifications "ResourceType=security-
group,Tags=[{Key=project,Value=$PROJECT_TAG},{Key=tutorial,Value=$TUTORIAL_TAG}]"
 \
        --query 'GroupId' --output text)

    if [[ -z "$SECURITY_GROUP_ID" ]]; then
        log "ERROR: Failed to create security group"
        exit 1
    fi

    # Add HTTP access rule for nginx web server with restricted CIDR
    # SECURITY FIX: Restrict access to specific CIDR if available, otherwise
 document the risk
    log "WARNING: Security group allows HTTP (port 80) from 0.0.0.0/0 - restrict
 this in production"
    aws ec2 authorize-security-group-ingress \
        --group-id "$SECURITY_GROUP_ID" \
```

```
        --protocol tcp \
        --port 80 \
        --cidr "0.0.0.0/0"

    log "Created security group: $SECURITY_GROUP_ID"
    log "Added HTTP access on port 80"
    CREATED_RESOURCES+=("Security Group: $SECURITY_GROUP_ID")
}

# Function to get ECS optimized AMI
get_ecs_ami() {
    log "Getting ECS-optimized AMI ID..."

    ECS_AMI_ID=$(aws ssm get-parameters \
        --names /aws/service/ecs/optimized-ami/amazon-linux-2/recommended \
        --query 'Parameters[0].Value' --output text | jq -r '.image_id')

    if [[ -z "$ECS_AMI_ID" ]]; then
        log "ERROR: Failed to get ECS-optimized AMI ID"
        exit 1
    fi

    log "ECS-optimized AMI ID: $ECS_AMI_ID"
}

# Function to create IAM role for ECS instance (if it doesn't exist)
ensure_ecs_instance_role() {
    log "Checking for ecsInstanceRole..."

    if ! aws iam get-role --role-name ecsInstanceRole &> /dev/null; then
        log "Creating ecsInstanceRole..."

        # Create trust policy
        cat > ecs-instance-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
```

```
  ]
}
EOF

        # SECURITY FIX: Validate JSON before using
        if ! jq empty ecs-instance-trust-policy.json 2>/dev/null; then
            log "ERROR: Invalid JSON in trust policy"
            rm -f ecs-instance-trust-policy.json
            exit 1
        fi

        # Create role
        aws iam create-role \
            --role-name ecsInstanceRole \
            --assume-role-policy-document file://ecs-instance-trust-policy.json

        aws iam tag-role --role-name ecsInstanceRole --tags Key=project,Value=
$PROJECT_TAG Key=tutorial,Value=$TUTORIAL_TAG

        # Attach managed policy
        aws iam attach-role-policy \
            --role-name ecsInstanceRole \
            --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role

        # Create instance profile
        aws iam create-instance-profile --instance-profile-name ecsInstanceRole

        # Add role to instance profile
        aws iam add-role-to-instance-profile \
            --instance-profile-name ecsInstanceRole \
            --role-name ecsInstanceRole

        # FIXED: Enhanced wait for role to be ready
        log "Waiting for IAM role to be ready..."
        aws iam wait role-exists --role-name ecsInstanceRole
        sleep 30  # Additional buffer for eventual consistency

        rm -f ecs-instance-trust-policy.json
        log "Created ecsInstanceRole"
        CREATED_RESOURCES+=("IAM Role: ecsInstanceRole")
    else
        log "ecsInstanceRole already exists"
    fi
```

```
}

# Function to launch container instance
launch_container_instance() {
    log "Launching ECS container instance..."

    # Create user data script
    cat > ecs-user-data.sh << EOF
#!/bin/bash
echo ECS_CLUSTER=$CLUSTER_NAME >> /etc/ecs/ecs.config
EOF

    # SECURITY FIX: Validate user data script before use
    if ! bash -n ecs-user-data.sh 2>/dev/null; then
        log "ERROR: Invalid user data script"
        rm -f ecs-user-data.sh
        exit 1
    fi

    INSTANCE_ID=$(aws ec2 run-instances \
        --image-id "$ECS_AMI_ID" \
        --instance-type t3.micro \
        --key-name "$KEY_PAIR_NAME" \
        --security-group-ids "$SECURITY_GROUP_ID" \
        --subnet-id "$DEFAULT_SUBNET" \
        --iam-instance-profile Name=ecsInstanceRole \
        --user-data file://ecs-user-data.sh \
        --tag-specifications "ResourceType=instance,Tags=[{Key=Name,Value=ecs-
tutorial-instance},{Key=project,Value=$PROJECT_TAG},{Key=tutorial,Value=
$TUTORIAL_TAG}]" \
        --monitoring Enabled=false \
        --metadata-options HttpTokens=required,HttpPutResponseHopLimit=1 \
        --query 'Instances[0].InstanceId' --output text)

    if [[ -z "$INSTANCE_ID" ]]; then
        log "ERROR: Failed to launch EC2 instance"
        rm -f ecs-user-data.sh
        exit 1
    fi

    log "Launched EC2 instance: $INSTANCE_ID"
    log "Instance metadata options: IMDSv2 enforced with hop limit 1"
    CREATED_RESOURCES+=("EC2 Instance: $INSTANCE_ID")
```

```
    # Wait for instance to be running
    log "Waiting for instance to be running..."
    aws ec2 wait instance-running --instance-ids "$INSTANCE_ID"

    # Wait for ECS agent to register
    log "Waiting for ECS agent to register with cluster..."
    local max_attempts=30
    local attempt=0

    while [[ $attempt -lt $max_attempts ]]; do
        CONTAINER_INSTANCES=$(aws ecs list-container-instances --cluster
 "$CLUSTER_NAME" --query 'containerInstanceArns' --output text)
        if [[ -n "$CONTAINER_INSTANCES" && "$CONTAINER_INSTANCES" != "None" ]];
 then
            log "Container instance registered successfully"
            break
        fi

        attempt=$((attempt + 1))
        log "Waiting for container instance registration... (attempt $attempt/
$max_attempts)"
        sleep 10
    done

    if [[ $attempt -eq $max_attempts ]]; then
        log "ERROR: Container instance failed to register within expected time"
        rm -f ecs-user-data.sh
        exit 1
    fi

    rm -f ecs-user-data.sh
}

# Function to register task definition
register_task_definition() {
    log "Creating task definition..."

    # Create nginx task definition JSON matching the tutorial
    cat > task-definition.json << 'EOF'
{
    "family": "TASK_FAMILY_PLACEHOLDER",
    "containerDefinitions": [
        {
            "name": "nginx",
```

```
            "image": "public.ecr.aws/docker/library/nginx:latest",
            "cpu": 256,
            "memory": 512,
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/nginx-task",
                    "awslogs-region": "REGION_PLACEHOLDER",
                    "awslogs-stream-prefix": "ecs"
                }
            }
        }
    ],
    "requiresCompatibilities": ["EC2"],
    "networkMode": "bridge",
    "tags": [
        {
            "key": "project",
            "value": "PROJECT_TAG_PLACEHOLDER"
        },
        {
            "key": "tutorial",
            "value": "TUTORIAL_TAG_PLACEHOLDER"
        }
    ]
}
EOF

    # Replace placeholders securely
    sed -i "s|TASK_FAMILY_PLACEHOLDER|$TASK_FAMILY|g" task-definition.json
    sed -i "s|REGION_PLACEHOLDER|$AWS_REGION|g" task-definition.json
    sed -i "s|PROJECT_TAG_PLACEHOLDER|$PROJECT_TAG|g" task-definition.json
    sed -i "s|TUTORIAL_TAG_PLACEHOLDER|$TUTORIAL_TAG|g" task-definition.json

    # FIXED: Validate JSON before registration
    if ! jq empty task-definition.json 2>/dev/null; then
```

```
        log "ERROR: Invalid JSON in task definition"
        rm -f task-definition.json
        exit 1
    fi

    TASK_DEFINITION_ARN=$(aws ecs register-task-definition \
        --cli-input-json file://task-definition.json \
        --query 'taskDefinition.taskDefinitionArn' --output text)

    if [[ -z "$TASK_DEFINITION_ARN" ]]; then
        log "ERROR: Failed to register task definition"
        rm -f task-definition.json
        exit 1
    fi

    log "Registered task definition: $TASK_DEFINITION_ARN"
    log "Task definition includes CloudWatch Logs configuration for monitoring"
    CREATED_RESOURCES+=("Task Definition: $TASK_DEFINITION_ARN")

    rm -f task-definition.json
}

# Function to create service
create_service() {
    log "Creating ECS service..."

    SERVICE_ARN=$(aws ecs create-service \
        --cluster "$CLUSTER_NAME" \
        --service-name "$SERVICE_NAME" \
        --task-definition "$TASK_FAMILY" \
        --desired-count 1 \
        --tags key=project,value=$PROJECT_TAG key=tutorial,value=$TUTORIAL_TAG \
        --query 'service.serviceArn' --output text)

    if [[ -z "$SERVICE_ARN" ]]; then
        log "ERROR: Failed to create service"
        exit 1
    fi

    log "Created service: $SERVICE_ARN"
    CREATED_RESOURCES+=("ECS Service: $SERVICE_NAME")

    # Wait for service to be stable
    log "Waiting for service to be stable..."
```

```
    aws ecs wait services-stable --cluster "$CLUSTER_NAME" --services
 "$SERVICE_NAME"

    log "Service is now stable and running"

    # Get the task ARN for monitoring
    TASK_ARN=$(aws ecs list-tasks --cluster "$CLUSTER_NAME" --service-name
 "$SERVICE_NAME" --query 'taskArns[0]' --output text)
    if [[ -n "$TASK_ARN" && "$TASK_ARN" != "None" ]]; then
        log "Service task: $TASK_ARN"
        CREATED_RESOURCES+=("ECS Task: $TASK_ARN")
    fi
}

# Function to demonstrate monitoring and testing
demonstrate_monitoring() {
    log "Demonstrating monitoring capabilities..."

    # List services
    log "Listing services in cluster:"
    aws ecs list-services --cluster "$CLUSTER_NAME" --output table

    # Describe service
    log "Service details:"
    aws ecs describe-services --cluster "$CLUSTER_NAME" --
services "$SERVICE_NAME" --output table --query 'services[0].
{ServiceName:serviceName,Status:status,RunningCount:runningCount,DesiredCount:desiredCoun

    # List tasks
    log "Listing tasks in service:"
    aws ecs list-tasks --cluster "$CLUSTER_NAME" --service-name "$SERVICE_NAME"
 --output table

    # Describe task
    if [[ -n "$TASK_ARN" && "$TASK_ARN" != "None" ]]; then
        log "Task details:"
        aws ecs describe-tasks --cluster "$CLUSTER_NAME"
 --tasks "$TASK_ARN" --output table --query 'tasks[0].
{TaskArn:taskArn,LastStatus:lastStatus,DesiredStatus:desiredStatus,CreatedAt:createdAt}'
    fi

    # List container instances
    log "Container instances in cluster:"
    aws ecs list-container-instances --cluster "$CLUSTER_NAME" --output table
```

```
    # Describe container instance
    CONTAINER_INSTANCE_ARN=$(aws ecs list-container-instances --cluster
 "$CLUSTER_NAME" --query 'containerInstanceArns[0]' --output text)
    if [[ -n "$CONTAINER_INSTANCE_ARN" && "$CONTAINER_INSTANCE_ARN" != "None" ]];
 then
        log "Container instance details:"
        aws ecs describe-container-instances --cluster "$CLUSTER_NAME"
 --container-instances "$CONTAINER_INSTANCE_ARN" --output table --query
 'containerInstances[0].
{Arn:containerInstanceArn,Status:status,RunningTasks:runningTasksCount,PendingTasks:pendi
    fi

    # Test the nginx web server
    log "Testing nginx web server..."
    PUBLIC_IP=$(aws ec2 describe-instances --instance-ids "$INSTANCE_ID" --query
 'Reservations[0].Instances[0].PublicIpAddress' --output text)

    if [[ -n "$PUBLIC_IP" && "$PUBLIC_IP" != "None" ]]; then
        log "Container instance public IP: $PUBLIC_IP"
        log "Testing HTTP connection to nginx..."

        # Wait a moment for nginx to be fully ready
        sleep 10

        if curl -s --connect-timeout 10 "http://$PUBLIC_IP" | grep -q "Welcome to
 nginx"; then
            log "SUCCESS: Nginx web server is responding correctly"
            echo ""
            echo "========================================="
            echo "WEB SERVER TEST SUCCESSFUL"
            echo "========================================="
            echo "You can access your nginx web server at: http://$PUBLIC_IP"
            echo "The nginx welcome page should be visible in your browser."
        else
            log "WARNING: Nginx web server may not be fully ready yet. Try
 accessing http://$PUBLIC_IP in a few minutes."
        fi
    else
        log "WARNING: Could not retrieve public IP address"
    fi
}

# Main execution
```

```
main() {
    log "Starting ECS EC2 Launch Type Tutorial (UPDATED VERSION)"
    log "Log file: $LOG_FILE"
    log "Security improvements: IMDSv2 enforced, JSON validation, input
 sanitization, CloudWatch Logs configured"

    check_prerequisites
    create_cluster
    create_key_pair
    create_security_group
    get_ecs_ami
    ensure_ecs_instance_role
    launch_container_instance
    register_task_definition
    create_service
    demonstrate_monitoring

    log "Tutorial completed successfully!"

    echo ""
    echo "========================================="
    echo "TUTORIAL COMPLETED SUCCESSFULLY"
    echo "========================================="
    echo "Resources created:"
    for resource in "${CREATED_RESOURCES[@]}"; do
        echo "  - $resource"
    done
    echo ""
    echo "The nginx service will continue running and maintain the desired task
 count."
    echo "You can monitor the service status using:"
    echo "  aws ecs describe-services --cluster $CLUSTER_NAME --services
 $SERVICE_NAME"
    echo ""
    if [[ -n "${PUBLIC_IP:-}" ]]; then
        echo "Access your web server at: http://$PUBLIC_IP"
        echo ""
    fi
    echo "========================================="
    echo "CLEANUP CONFIRMATION"
    echo "========================================="
    log "Auto-confirming cleanup - proceeding with resource cleanup"
    cleanup_resources
    log "All resources have been cleaned up"
```

```
}

# Run main function
main "$@"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddRoleToInstanceProfile
  - AttachRolePolicy
  - AuthorizeSecurityGroupIngress
  - CreateCluster
  - CreateInstanceProfile
  - CreateKeyPair
  - CreateRole
  - CreateSecurityGroup
  - CreateService
  - DeleteCluster
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DeleteService
  - DeregisterTaskDefinition
  - DescribeContainerInstances
  - DescribeInstances
  - DescribeServices
  - DescribeSubnets
  - DescribeTasks
  - DescribeVpcs
  - GetCallerIdentity
  - GetParameters
  - GetRole
  - ListContainerInstances
  - ListServices
  - ListTasks

- RegisterTaskDefinition
- RunInstances
- StopTask
- TerminateInstances
- UpdateService
- Wait (Amazon EC2)
- Wait (Amazon ECS)
- Wait (IAM)

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Creating a managed monitoring workspace

The following code example shows how to:

- Create an IAM role for your workspace
- Create a Grafana workspace
- Configure authentication
- Configure optional settings
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash
```

```
# Amazon Managed Grafana Workspace Creation Script
# This script creates an Amazon Managed Grafana workspace and configures it

# Set up logging
LOG_FILE="grafana-workspace-creation.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon Managed Grafana workspace creation script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2

    if echo "$output" | grep -i "error\|exception\|fail" > /dev/null; then
        echo "ERROR: Command '$cmd' failed with output:"
        echo "$output"
        cleanup_on_error
        exit 1
    fi
}

# Function to clean up resources on error
cleanup_on_error() {
    echo "Error encountered. Attempting to clean up resources..."

    if [ -n "$WORKSPACE_ID" ]; then
        echo "Deleting workspace $WORKSPACE_ID..."
        aws grafana delete-workspace --workspace-id "$WORKSPACE_ID"
    fi

    if [ -n "$ROLE_NAME" ]; then
        echo "Detaching policies from role $ROLE_NAME..."
        if [ -n "$POLICY_ARN" ]; then
            aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$POLICY_ARN"
        fi

        echo "Deleting role $ROLE_NAME..."
        aws iam delete-role --role-name "$ROLE_NAME"
    fi

    if [ -n "$POLICY_ARN" ]; then
```

```
        echo "Deleting policy..."
        aws iam delete-policy --policy-arn "$POLICY_ARN"
    fi

    # Clean up JSON files
    rm -f trust-policy.json cloudwatch-policy.json

    echo "Cleanup completed. See $LOG_FILE for details."
}

# Generate a random identifier for resource names
RANDOM_ID=$(openssl rand -hex 4)
WORKSPACE_NAME="GrafanaWorkspace-${RANDOM_ID}"
ROLE_NAME="GrafanaWorkspaceRole-${RANDOM_ID}"

echo "Using workspace name: $WORKSPACE_NAME"
echo "Using role name: $ROLE_NAME"

# Step 1: Get AWS account ID
echo "Getting AWS account ID..."
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
check_error "$ACCOUNT_ID" "get-caller-identity"
echo "AWS Account ID: $ACCOUNT_ID"

# Step 2: Create IAM role for Grafana workspace
echo "Creating IAM role for Grafana workspace..."

# Create trust policy document
cat > trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "grafana.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role
```

```
ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$ROLE_NAME" \
  --assume-role-policy-document file://trust-policy.json \
  --description "Role for Amazon Managed Grafana workspace")

check_error "$ROLE_OUTPUT" "create-role"
aws iam tag-role --role-name "$ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=amazon-managed-grafana-gs
echo "IAM role created successfully"

# Extract role ARN
ROLE_ARN=$(echo "$ROLE_OUTPUT" | grep -o '"Arn": "[^"]*' | cut -d'"' -f4)
echo "Role ARN: $ROLE_ARN"

# Attach policies to the role
echo "Attaching policies to the role..."

# CloudWatch policy
cat > cloudwatch-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmsForMetric",
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:ListMetrics",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:GetMetricData"
      ],
      "Resource": "*"
    }
  ]
}
EOF

POLICY_OUTPUT=$(aws iam create-policy \
  --policy-name "GrafanaCloudWatchPolicy-${RANDOM_ID}" \
  --policy-document file://cloudwatch-policy.json)

check_error "$POLICY_OUTPUT" "create-policy"
```

```
POLICY_ARN=$(echo "$POLICY_OUTPUT" | grep -o '"Arn": "[^"]*' | cut -d'"' -f4)
echo "CloudWatch policy ARN: $POLICY_ARN"

ATTACH_OUTPUT=$(aws iam attach-role-policy \
  --role-name "$ROLE_NAME" \
  --policy-arn "$POLICY_ARN")

check_error "$ATTACH_OUTPUT" "attach-role-policy"
echo "CloudWatch policy attached to role"

# Step 3: Create the Grafana workspace
echo "Creating Amazon Managed Grafana workspace..."
WORKSPACE_OUTPUT=$(aws grafana create-workspace \
  --workspace-name "$WORKSPACE_NAME" \
  --authentication-providers "SAML" \
  --permission-type "CUSTOMER_MANAGED" \
  --account-access-type "CURRENT_ACCOUNT" \
  --workspace-role-arn "$ROLE_ARN" \
  --workspace-data-sources "CLOUDWATCH" "PROMETHEUS" "XRAY" \
  --grafana-version "10.4" \
  --tags Environment=Development,project=doc-smith,tutorial=amazon-managed-
grafana-gs)

check_error "$WORKSPACE_OUTPUT" "create-workspace"

echo "Workspace creation initiated:"
echo "$WORKSPACE_OUTPUT"

# Extract workspace ID
WORKSPACE_ID=$(echo "$WORKSPACE_OUTPUT" | grep -o '"id": "[^"]*' | cut -d'"' -f4)

if [ -z "$WORKSPACE_ID" ]; then
    echo "ERROR: Failed to extract workspace ID from output"
    exit 1
fi

echo "Workspace ID: $WORKSPACE_ID"

# Step 4: Wait for workspace to become active
echo "Waiting for workspace to become active. This may take several minutes..."
ACTIVE=false
MAX_ATTEMPTS=30
ATTEMPT=0
```

```
while [ $ACTIVE = false ] && [ $ATTEMPT -lt $MAX_ATTEMPTS ]; do
    ATTEMPT=$((ATTEMPT+1))
    echo "Checking workspace status (attempt $ATTEMPT of $MAX_ATTEMPTS)..."

    DESCRIBE_OUTPUT=$(aws grafana describe-workspace --workspace-id
 "$WORKSPACE_ID")
    check_error "$DESCRIBE_OUTPUT" "describe-workspace"

    STATUS=$(echo "$DESCRIBE_OUTPUT" | grep -o '"status": "[^"]*' | cut -d'"' -
f4)
    echo "Current status: $STATUS"

    if [ "$STATUS" = "ACTIVE" ]; then
        ACTIVE=true
        echo "Workspace is now ACTIVE"
    elif [ "$STATUS" = "FAILED" ]; then
        echo "ERROR: Workspace creation failed"
        cleanup_on_error
        exit 1
    else
        echo "Workspace is still being created. Waiting 30 seconds..."
        sleep 30
    fi
done

if [ $ACTIVE = false ]; then
    echo "ERROR: Workspace did not become active within the expected time"
    cleanup_on_error
    exit 1
fi

# Extract workspace endpoint URL
WORKSPACE_URL=$(echo "$DESCRIBE_OUTPUT" | grep -o '"endpoint": "[^"]*' | cut -
d'"' -f4)
echo "Workspace URL: https://$WORKSPACE_URL"

# Step 5: Display workspace information
echo ""
echo "========================================="
echo "WORKSPACE INFORMATION"
echo "========================================="
echo "Workspace ID: $WORKSPACE_ID"
echo "Workspace URL: https://$WORKSPACE_URL"
echo "Workspace Name: $WORKSPACE_NAME"
```

```
echo "IAM Role: $ROLE_NAME"
echo ""
echo "Note: Since SAML authentication is used, you need to configure SAML
 settings"
echo "using the AWS Management Console or the update-workspace-authentication
 command."
echo "========================================="

# Step 6: Prompt for cleanup
echo ""
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Resources created:"
echo "- Amazon Managed Grafana workspace: $WORKSPACE_ID"
echo "- IAM Role: $ROLE_NAME"
echo "- IAM Policy: GrafanaCloudWatchPolicy-${RANDOM_ID}"
echo ""
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "$CLEANUP_CHOICE" =~ ^[Yy] ]]; then
    echo "Cleaning up resources..."

    echo "Deleting workspace $WORKSPACE_ID..."
    DELETE_OUTPUT=$(aws grafana delete-workspace --workspace-id "$WORKSPACE_ID")
    check_error "$DELETE_OUTPUT" "delete-workspace"

    echo "Waiting for workspace to be deleted..."
    DELETED=false
    ATTEMPT=0

    while [ $DELETED = false ] && [ $ATTEMPT -lt $MAX_ATTEMPTS ]; do
        ATTEMPT=$((ATTEMPT+1))
        echo "Checking deletion status (attempt $ATTEMPT of $MAX_ATTEMPTS)..."

        if aws grafana describe-workspace --workspace-id "$WORKSPACE_ID" 2>&1 |
 grep -i "not found\|does not exist" > /dev/null; then
            DELETED=true
            echo "Workspace has been deleted"
        else
            echo "Workspace is still being deleted. Waiting 30 seconds..."
            sleep 30
        fi
```

```
    done

    if [ $DELETED = false ]; then
        echo "WARNING: Workspace deletion is taking longer than expected. It may
 still be in progress."
    fi

    # Detach policy from role
    echo "Detaching policy from role..."
    aws iam detach-role-policy \
      --role-name "$ROLE_NAME" \
      --policy-arn "$POLICY_ARN"

    # Delete policy
    echo "Deleting IAM policy..."
    aws iam delete-policy \
      --policy-arn "$POLICY_ARN"

    # Delete role
    echo "Deleting IAM role..."
    aws iam delete-role \
      --role-name "$ROLE_NAME"

    # Clean up JSON files
    rm -f trust-policy.json cloudwatch-policy.json

    echo "Cleanup completed"
else
    echo "Skipping cleanup. Resources will remain in your AWS account."
fi

echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreatePolicy
  - CreateRole
  - CreateWorkspace
  - DeletePolicy
  - DeleteRole

- DeleteWorkspace
- DescribeWorkspace
- DetachRolePolicy
- GetCallerIdentity

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Creating a monitoring dashboard with function name as a variable

The following code example shows how to:

- Create a CloudWatch dashboard
- Add Lambda metrics widgets with a function name variable
- Verify the dashboard
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Script to create a CloudWatch dashboard with Lambda function name as a variable
# This script creates a CloudWatch dashboard that allows you to switch between
 different Lambda functions

# Set up logging with secure permissions
LOG_FILE="${HOME}/.cloudwatch-dashboard-script.log"
touch "$LOG_FILE" && chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1
```

```
echo "$(date): Starting CloudWatch dashboard creation script"

# Security: Set strict error handling
set -uo pipefail
trap 'handle_error "Script failed at line $LINENO"' ERR

# Function to handle errors
handle_error() {
    local error_msg="${1:-Unknown error}"
    echo "ERROR: $error_msg" >&2
    echo "Resources created:"
    echo "- CloudWatch Dashboard: LambdaMetricsDashboard"
    echo ""
    echo "=========================================="
    echo "CLEANUP CONFIRMATION"
    echo "=========================================="
    echo "An error occurred. Proceeding with automatic cleanup..."

    echo "Cleaning up resources..."
    aws cloudwatch delete-dashboards --dashboard-names LambdaMetricsDashboard 2>/
dev/null || true

    # Clean up temporary files
    if [ -n "${TEMP_DIR:-}" ] && [ -d "$TEMP_DIR" ]; then
        rm -rf "$TEMP_DIR"
    fi
    rm -f dashboard-body.json

    echo "Cleanup complete."
    exit 1
}

# Security: Validate AWS CLI is installed
if ! command -v aws &> /dev/null; then
    handle_error "AWS CLI is not installed. Please install it and try again."
fi

# Check if AWS CLI is installed and configured
echo "Checking AWS CLI configuration..."
if ! aws sts get-caller-identity > /dev/null 2>&1; then
    handle_error "AWS CLI is not properly configured. Please configure it with
 'aws configure' and try again."
fi
```

```
# Get the current region securely
REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$REGION" ]; then
    REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $REGION"
fi
echo "Using region: $REGION"

# Validate region format
if ! [[ "$REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]{1}$ ]]; then
    handle_error "Invalid AWS region format: $REGION"
fi

# Check if there are any Lambda functions in the account
echo "Checking for Lambda functions..."
LAMBDA_FUNCTIONS=$(aws lambda list-functions --region "$REGION" --query
 "Functions[*].FunctionName" --output text 2>/dev/null || echo "")

if [ -z "$LAMBDA_FUNCTIONS" ]; then
    echo "No Lambda functions found in your account. Creating a simple test
 function..."

    # Create a temporary directory for Lambda function code with secure
 permissions
    TEMP_DIR=$(mktemp -d)
    chmod 700 "$TEMP_DIR"
    trap 'rm -rf "$TEMP_DIR"' EXIT

    # Create a simple Lambda function
    cat > "$TEMP_DIR/index.js" << 'EOF'
exports.handler = async (event) => {
    console.log('Event:', JSON.stringify(event, null, 2));
    return {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
};
EOF

    # Zip the function code
    if ! cd "$TEMP_DIR"; then
        handle_error "Failed to change to temporary directory"
    fi
```

```
    if ! zip -q function.zip index.js; then
        handle_error "Failed to create zip file"
    fi

    # Create a role for the Lambda function with restricted trust policy
    ROLE_NAME="LambdaDashboardTestRole-$(date +%s)"
    TRUST_POLICY='{"Version":"2012-10-17",       "Statement":
[{"Effect":"Allow","Principal":
{"Service":"lambda.amazonaws.com"},"Action":"sts:AssumeRole"}]}'

    if ! ROLE_ARN=$(aws iam create-role \
        --role-name "$ROLE_NAME" \
        --assume-role-policy-document "$TRUST_POLICY" \
        --tags Key=project,Value=doc-smith Key=tutorial,Value=cloudwatch-
dynamicdash \
        --query "Role.Arn" \
        --output text 2>/dev/null); then
        handle_error "Failed to create IAM role for Lambda function"
    fi

    echo "Waiting for role to be available..."
    sleep 10

    # Attach basic Lambda execution policy
    if ! aws iam attach-role-policy \
        --role-name "$ROLE_NAME" \
        --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"; then
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to attach policy to IAM role"
    fi

    # Create the Lambda function
    FUNCTION_NAME="DashboardTestFunction-$(date +%s)"
    if ! aws lambda create-function \
        --function-name "$FUNCTION_NAME" \
        --runtime nodejs18.x \
        --role "$ROLE_ARN" \
        --handler index.handler \
        --zip-file fileb://function.zip \
        --tags project=doc-smith,tutorial=cloudwatch-dynamicdash \
        --region "$REGION" > /dev/null 2>&1; then
        aws iam detach-role-policy \
```

```
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to create Lambda function"
    fi

    # Invoke the function to generate some metrics
    echo "Invoking Lambda function to generate metrics..."
    for i in {1..5}; do
        aws lambda invoke --function-name "$FUNCTION_NAME" --payload '{}' /dev/
null --region "$REGION" > /dev/null 2>&1 || true
        sleep 1
    done

    # Go back to original directory
    cd - > /dev/null

    # Set the function name for the dashboard
    DEFAULT_FUNCTION="$FUNCTION_NAME"
else
    # Use the first Lambda function as default
    DEFAULT_FUNCTION=$(echo "$LAMBDA_FUNCTIONS" | awk '{print $1}')
    echo "Found Lambda functions. Using $DEFAULT_FUNCTION as default."
    FUNCTION_NAME=""
    ROLE_NAME=""
fi

# Create a dashboard with Lambda metrics and a function name variable
echo "Creating CloudWatch dashboard with Lambda function name variable..."

# Create a JSON file for the dashboard body with secure permissions
DASHBOARD_JSON="dashboard-body-$$.json"
touch "$DASHBOARD_JSON" && chmod 600 "$DASHBOARD_JSON"

# Escape special characters in region and function name for JSON
REGION_ESCAPED=$(printf '%s\n' "$REGION" | sed 's:[\/&]:\\&:g')
FUNCTION_ESCAPED=$(printf '%s\n' "$DEFAULT_FUNCTION" | sed 's:[\/&]:\\&:g')

cat > "$DASHBOARD_JSON" << EOF
{
  "widgets": [
    {
      "type": "metric",
```

```
      "x": 0,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "\${FunctionName}" ],
          [ ".", "Errors", ".", "." ],
          [ ".", "Throttles", ".", "." ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Lambda Function Metrics for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 0,
      "y": 6,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Duration", "FunctionName", "\${FunctionName}",
{ "stat": "Average" } ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Duration for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 12,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
```

```
          [ "AWS/Lambda", "ConcurrentExecutions", "FunctionName", "\
${FunctionName}" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Concurrent Executions for \${FunctionName}",
        "period": 300
      }
    }
  ],
  "periodOverride": "auto",
  "variables": [
    {
      "type": "property",
      "id": "FunctionName",
      "property": "FunctionName",
      "label": "Lambda Function",
      "inputType": "select",
      "values": [
        {
          "value": "$FUNCTION_ESCAPED",
          "label": "$FUNCTION_ESCAPED"
        }
      ]
    }
  ]
}
EOF

# Validate JSON before sending
if ! jq empty "$DASHBOARD_JSON" 2>/dev/null; then
    handle_error "Invalid JSON generated for dashboard"
fi

# Create the dashboard using the JSON file
DASHBOARD_NAME="LambdaMetricsDashboard-$(date +%s)"
if ! DASHBOARD_RESULT=$(aws cloudwatch put-dashboard \
    --dashboard-name "$DASHBOARD_NAME" \
    --dashboard-body file://"$DASHBOARD_JSON" \
    --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
```

```
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to create CloudWatch dashboard."
fi

# Display any validation messages but continue
if echo "$DASHBOARD_RESULT" | grep -q "DashboardValidationMessages"; then
    echo "Dashboard created with validation messages:"
    echo "$DASHBOARD_RESULT"
    echo "These validation messages are warnings and the dashboard should still
 function."
else
    echo "Dashboard created successfully!"
fi

# Verify the dashboard was created
echo "Verifying dashboard creation..."
if ! DASHBOARD_INFO=$(aws cloudwatch get-dashboard --dashboard-name
 "$DASHBOARD_NAME" --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to verify dashboard creation."
fi

echo "Dashboard verification successful!"
echo "Dashboard details:"
echo "$DASHBOARD_INFO"

# List all dashboards to confirm
echo "Listing all dashboards:"
```

```
if ! DASHBOARDS=$(aws cloudwatch list-dashboards --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to list dashboards."
fi
echo "$DASHBOARDS"

# Show instructions for accessing the dashboard
echo ""
echo "Dashboard created successfully! To access it:"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/"
echo "2. In the navigation pane, choose Dashboards"
echo "3. Select $DASHBOARD_NAME"
echo "4. You should see a dropdown menu labeled 'Lambda Function' at the top of
 the dashboard"
echo "5. Use this dropdown to select different Lambda functions and see their
 metrics"
echo ""

# Create a list of resources for cleanup
RESOURCES=("- CloudWatch Dashboard: $DASHBOARD_NAME")
if [ -n "${FUNCTION_NAME:-}" ]; then
    RESOURCES+=("- Lambda Function: $FUNCTION_NAME")
    RESOURCES+=("- IAM Role: $ROLE_NAME")
fi

# Prompt for cleanup with automatic yes
echo "==========================================="
echo "CLEANUP CONFIRMATION"
echo "==========================================="
echo "Resources created:"
for resource in "${RESOURCES[@]}"; do
    echo "$resource"
done
echo ""
```

```
echo "Proceeding with automatic cleanup..."

CLEANUP_CHOICE="y"

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    echo "Cleaning up resources..."

    # Delete the dashboard
    if aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" --
region "$REGION" 2>/dev/null; then
        echo "Dashboard deleted successfully."
    else
        echo "WARNING: Failed to delete dashboard. You may need to delete it
 manually."
    fi

    # If we created a Lambda function, delete it and its role
    if [ -n "${FUNCTION_NAME:-}" ]; then
        echo "Deleting Lambda function..."
        if aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null; then
            echo "Lambda function deleted successfully."
        else
            echo "WARNING: Failed to delete Lambda function. You may need to
 delete it manually."
        fi

        echo "Detaching role policy..."
        if aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null; then
            echo "Role policy detached successfully."
        else
            echo "WARNING: Failed to detach role policy. You may need to detach
 it manually."
        fi

        echo "Deleting IAM role..."
        if aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null; then
            echo "IAM role deleted successfully."
        else
            echo "WARNING: Failed to delete IAM role. You may need to delete it
 manually."
```

```
        fi
    fi

    # Clean up the JSON file
    rm -f "$DASHBOARD_JSON"

    echo "Cleanup complete."
fi

echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateFunction
  - CreateRole
  - DeleteDashboards
  - DeleteFunction
  - DeleteRole
  - DetachRolePolicy
  - GetCallerIdentity
  - GetDashboard
  - Invoke
  - ListDashboards
  - ListFunctions
  - PutDashboard

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get a session token that requires an MFA token with AWS STS using an AWS SDK

The following code example shows how to get a session token that requires an MFA token.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create an IAM role that grants permission to list Amazon S3 buckets.
- Create an IAM user that has permission to assume the role only when MFA credentials are provided.
- Register an MFA device for the user.
- Provide MFA credentials to get a session token and use temporary credentials to list S3 buckets.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an IAM user, register an MFA device, and create a role that grants permission to let the user list S3 buckets only when MFA credentials are used.

```
def setup(iam_resource):
    """
    Creates a new user with no permissions.
    Creates a new virtual multi-factor authentication (MFA) device.
    Displays the QR code to seed the device.
    Asks for two codes from the MFA device.
    Registers the MFA device for the user.
    Creates an access key pair for the user.
    Creates an inline policy for the user that lets the user list Amazon S3
 buckets,
    but only when MFA credentials are used.

    Any MFA device that can scan a QR code will work with this demonstration.
```

```
    Common choices are mobile apps like LastPass Authenticator,
    Microsoft Authenticator, or Google Authenticator.

    :param iam_resource: A Boto3 AWS Identity and Access Management (IAM)
resource
                         that has permissions to create users, MFA devices, and
                         policies in the account.
    :return: The newly created user, user key, and virtual MFA device.
    """
    user = iam_resource.create_user(UserName=unique_name("user"))
    print(f"Created user {user.name}.")

    virtual_mfa_device = iam_resource.create_virtual_mfa_device(
        VirtualMFADeviceName=unique_name("mfa")
    )
    print(f"Created virtual MFA device {virtual_mfa_device.serial_number}")

    print(
        f"Showing the QR code for the device. Scan this in the MFA app of your "
        f"choice."
    )
    with open("qr.png", "wb") as qr_file:
        qr_file.write(virtual_mfa_device.qr_code_png)
    webbrowser.open(qr_file.name)

    print(f"Enter two consecutive code from your MFA device.")
    mfa_code_1 = input("Enter the first code: ")
    mfa_code_2 = input("Enter the second code: ")
    user.enable_mfa(
        SerialNumber=virtual_mfa_device.serial_number,
        AuthenticationCode1=mfa_code_1,
        AuthenticationCode2=mfa_code_2,
    )
    os.remove(qr_file.name)
    print(f"MFA device is registered with the user.")

    user_key = user.create_access_key_pair()
    print(f"Created access key pair for user.")

    print(f"Wait for user to be ready.", end="")
    progress_bar(10)

    user.create_policy(
        PolicyName=unique_name("user-policy"),
```

```
        PolicyDocument=json.dumps(
            {
                "Version":"2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Action": "s3:ListAllMyBuckets",
                        "Resource": "arn:aws:s3:::*",
                        "Condition": {"Bool": {"aws:MultiFactorAuthPresent":
 True}},
                    }
                ],
            }
        ),
    )
    print(
        f"Created an inline policy for {user.name} that lets the user list
 buckets, "
        f"but only when MFA credentials are present."
    )

    print("Give AWS time to propagate these new resources and connections.",
 end="")
    progress_bar(10)

    return user, user_key, virtual_mfa_device
```

Get temporary session credentials by passing an MFA token, and use the credentials to list S3 buckets for the account.

```
def list_buckets_with_session_token_with_mfa(mfa_serial_number, mfa_totp,
 sts_client):
    """
    Gets a session token with MFA credentials and uses the temporary session
    credentials to list Amazon S3 buckets.

    Requires an MFA device serial number and token.

    :param mfa_serial_number: The serial number of the MFA device. For a virtual
 MFA
```

```
                                device, this is an Amazon Resource Name (ARN).
    :param mfa_totp: A time-based, one-time password issued by the MFA device.
    :param sts_client: A Boto3 STS instance that has permission to assume the
 role.
    """
    if mfa_serial_number is not None:
        response = sts_client.get_session_token(
            SerialNumber=mfa_serial_number, TokenCode=mfa_totp
        )
    else:
        response = sts_client.get_session_token()
    temp_credentials = response["Credentials"]

    s3_resource = boto3.resource(
        "s3",
        aws_access_key_id=temp_credentials["AccessKeyId"],
        aws_secret_access_key=temp_credentials["SecretAccessKey"],
        aws_session_token=temp_credentials["SessionToken"],
    )

    print(f"Buckets for the account:")
    for bucket in s3_resource.buckets.all():
        print(bucket.name)
```

Destroy the resources created for the demo.

```
def teardown(user, virtual_mfa_device):
    """
    Removes all resources created during setup.

    :param user: The demo user.
    :param role: The demo MFA device.
    """
    for user_pol in user.policies.all():
        user_pol.delete()
        print("Deleted inline user policy.")
    for key in user.access_keys.all():
        key.delete()
        print("Deleted user's access key.")
    for mfa in user.mfa_devices.all():
```

```
        mfa.disassociate()
    virtual_mfa_device.delete()
    user.delete()
    print(f"Deleted {user.name}.")
```

Run this scenario by using the previously defined functions.

```
def usage_demo():
    """Drives the demonstration."""
    print("-" * 88)
    print(
        f"Welcome to the AWS Security Token Service assume role demo, "
        f"starring multi-factor authentication (MFA)!"
    )
    print("-" * 88)
    iam_resource = boto3.resource("iam")
    user, user_key, virtual_mfa_device = setup(iam_resource)
    try:
        sts_client = boto3.client(
            "sts", aws_access_key_id=user_key.id,
 aws_secret_access_key=user_key.secret
        )
        try:
            print("Listing buckets without specifying MFA credentials.")
            list_buckets_with_session_token_with_mfa(None, None, sts_client)
        except ClientError as error:
            if error.response["Error"]["Code"] == "AccessDenied":
                print("Got expected AccessDenied error.")
        mfa_totp = input("Enter the code from your registered MFA device: ")
        list_buckets_with_session_token_with_mfa(
            virtual_mfa_device.serial_number, mfa_totp, sts_client
        )
    finally:
        teardown(user, virtual_mfa_device)
        print("Thanks for watching!")
```

- For API details, see GetSessionToken in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with configuration management

The following code example shows how to:

- Create an Amazon S3 bucket
- Create an Amazon SNS topic
- Create an IAM role for Config
- Set up the Config configuration recorder
- Set up the Config delivery channel
- Start the configuration recorder
- Verify the Config setup

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS Config Setup Script (v2)
# This script sets up AWS Config with the AWS CLI

# Error handling
set -e
LOGFILE="aws-config-setup-v2.log"
touch $LOGFILE
exec > >(tee -a $LOGFILE)
exec 2>&1

# Function to handle errors
```

```
handle_error() {
    echo "ERROR: An error occurred at line $1"
    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Set trap for error handling
trap 'handle_error $LINENO' ERR

# Function to generate random identifier
generate_random_id() {
    echo $(openssl rand -hex 6)
}

# Function to check if command was successful
check_command() {
    if echo "$1" | grep -i "error" > /dev/null; then
        echo "ERROR: $1"
        return 1
    fi
    return 0
}

# Function to clean up resources
cleanup_resources() {
    if [ -n "$CONFIG_RECORDER_NAME" ]; then
        echo "Stopping configuration recorder..."
        aws configservice stop-configuration-recorder --configuration-recorder-
name "$CONFIG_RECORDER_NAME" 2>/dev/null || true
    fi

    # Check if we created a new delivery channel before trying to delete it
    if [ -n "$DELIVERY_CHANNEL_NAME" ] && [ "$CREATED_NEW_DELIVERY_CHANNEL" =
 "true" ]; then
        echo "Deleting delivery channel..."
        aws configservice delete-delivery-channel --delivery-channel-name
 "$DELIVERY_CHANNEL_NAME" 2>/dev/null || true
    fi

    if [ -n "$CONFIG_RECORDER_NAME" ] && [ "$CREATED_NEW_CONFIG_RECORDER" =
 "true" ]; then
        echo "Deleting configuration recorder..."
```

```
        aws configservice delete-configuration-recorder --configuration-recorder-
name "$CONFIG_RECORDER_NAME" 2>/dev/null || true
    fi

    if [ -n "$ROLE_NAME" ]; then
        if [ -n "$POLICY_NAME" ]; then
            echo "Detaching custom policy from role..."
            aws iam delete-role-policy --role-name "$ROLE_NAME" --policy-name
 "$POLICY_NAME" 2>/dev/null || true
        fi

        if [ -n "$MANAGED_POLICY_ARN" ]; then
            echo "Detaching managed policy from role..."
            aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$MANAGED_POLICY_ARN" 2>/dev/null || true
        fi

        echo "Deleting IAM role..."
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi

    if [ -n "$SNS_TOPIC_ARN" ]; then
        echo "Deleting SNS topic..."
        aws sns delete-topic --topic-arn "$SNS_TOPIC_ARN" 2>/dev/null || true
    fi

    if [ -n "$S3_BUCKET_NAME" ]; then
        echo "Emptying S3 bucket..."
        aws s3 rm "s3://$S3_BUCKET_NAME" --recursive 2>/dev/null || true

        echo "Deleting S3 bucket..."
        if [ "$BUCKET_IS_SHARED" = "false" ]; then
            aws s3api delete-bucket --bucket "$S3_BUCKET_NAME" 2>/dev/null ||
 true
        fi
    fi
}

# Function to display created resources
display_resources() {
    echo ""
    echo "========================================="
    echo "CREATED RESOURCES"
    echo "========================================="
```

```
    echo "S3 Bucket: $S3_BUCKET_NAME"
    echo "SNS Topic ARN: $SNS_TOPIC_ARN"
    echo "IAM Role: $ROLE_NAME"
    if [ "$CREATED_NEW_CONFIG_RECORDER" = "true" ]; then
        echo "Configuration Recorder: $CONFIG_RECORDER_NAME (newly created)"
    else
        echo "Configuration Recorder: $CONFIG_RECORDER_NAME (existing)"
    fi
    if [ "$CREATED_NEW_DELIVERY_CHANNEL" = "true" ]; then
        echo "Delivery Channel: $DELIVERY_CHANNEL_NAME (newly created)"
    else
        echo "Delivery Channel: $DELIVERY_CHANNEL_NAME (existing)"
    fi
    echo "============================================="
}

# Get AWS account ID
echo "Getting AWS account ID..."
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
if [ -z "$ACCOUNT_ID" ]; then
    echo "ERROR: Failed to get AWS account ID"
    exit 1
fi
echo "AWS Account ID: $ACCOUNT_ID"

# Generate random identifier for resources
RANDOM_ID=$(generate_random_id)
echo "Generated random identifier: $RANDOM_ID"

# Step 1: Create an S3 bucket
# Check for shared prereq bucket
PREREQ_BUCKET=$(aws cloudformation describe-stacks --stack-name tutorial-prereqs-
bucket \
    --query 'Stacks[0].Outputs[?OutputKey==`BucketName`].OutputValue' --output
 text 2>/dev/null)
if [ -n "$PREREQ_BUCKET" ] && [ "$PREREQ_BUCKET" != "None" ]; then
    S3_BUCKET_NAME="$PREREQ_BUCKET"
    BUCKET_IS_SHARED=true
    echo "Using shared bucket: $S3_BUCKET_NAME"
else
    BUCKET_IS_SHARED=false
    S3_BUCKET_NAME="configservice-${RANDOM_ID}"
    echo "Creating S3 bucket: $S3_BUCKET_NAME"
fi
```

```
# Get the current region
AWS_REGION=$(aws configure get region)
if [ -z "$AWS_REGION" ]; then
    AWS_REGION="us-east-1"  # Default to us-east-1 if no region is configured
fi
echo "Using AWS Region: $AWS_REGION"

# Create bucket with appropriate command based on region
if [ "$BUCKET_IS_SHARED" = "false" ]; then
    if [ "$AWS_REGION" = "us-east-1" ]; then
        BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME")
    else
        BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME" --
create-bucket-configuration LocationConstraint="$AWS_REGION")
    fi
    check_command "$BUCKET_RESULT"
    echo "S3 bucket created: $S3_BUCKET_NAME"

    aws s3api put-bucket-tagging --bucket "$S3_BUCKET_NAME" --tagging
 'TagSet=[{Key=project,Value=doc-smith},{Key=tutorial,Value=aws-config-gs}]'
    echo "Tags applied to S3 bucket"
else
    echo "Using shared bucket: $S3_BUCKET_NAME (skipping creation)"
fi

# Block public access for the bucket
aws s3api put-public-access-block \
    --bucket "$S3_BUCKET_NAME" \
    --public-access-block-configuration
 "BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets
echo "Public access blocked for bucket"

# Step 2: Create an SNS topic
TOPIC_NAME="config-topic-${RANDOM_ID}"
echo "Creating SNS topic: $TOPIC_NAME"
SNS_RESULT=$(aws sns create-topic --name "$TOPIC_NAME" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-config-gs)
check_command "$SNS_RESULT"
SNS_TOPIC_ARN=$(echo "$SNS_RESULT" | grep -o 'arn:aws:sns:[^"]*')
echo "SNS topic created: $SNS_TOPIC_ARN"

# Step 3: Create an IAM role for AWS Config
ROLE_NAME="config-role-${RANDOM_ID}"
```

```
POLICY_NAME="config-delivery-permissions"
MANAGED_POLICY_ARN="arn:aws:iam::aws:policy/service-role/AWS_ConfigRole"

echo "Creating trust policy document..."
cat > config-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "config.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

echo "Creating IAM role: $ROLE_NAME"
ROLE_RESULT=$(aws iam create-role --role-name "$ROLE_NAME" --assume-role-policy-
document file://config-trust-policy.json)
check_command "$ROLE_RESULT"
ROLE_ARN=$(echo "$ROLE_RESULT" | grep -o 'arn:aws:iam::[^"]*' | head -1)
echo "IAM role created: $ROLE_ARN"

aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-config-gs
echo "Tags applied to IAM role"

echo "Attaching AWS managed policy to role..."
ATTACH_RESULT=$(aws iam attach-role-policy --role-name "$ROLE_NAME" --policy-arn
 "$MANAGED_POLICY_ARN")
check_command "$ATTACH_RESULT"
echo "AWS managed policy attached"

echo "Creating custom policy document for S3 and SNS access..."
cat > config-delivery-permissions.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::${S3_BUCKET_NAME}/AWSLogs/${ACCOUNT_ID}/*",
      "Condition": {
        "StringLike": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketAcl"
      ],
      "Resource": "arn:aws:s3:::${S3_BUCKET_NAME}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": "${SNS_TOPIC_ARN}"
    }
  ]
}
EOF

echo "Attaching custom policy to role..."
POLICY_RESULT=$(aws iam put-role-policy --role-name "$ROLE_NAME" --policy-name
 "$POLICY_NAME" --policy-document file://config-delivery-permissions.json)
check_command "$POLICY_RESULT"
echo "Custom policy attached"

# Wait for IAM role to propagate
echo "Waiting for IAM role to propagate (15 seconds)..."
sleep 15

# Step 4: Check if configuration recorder already exists
CONFIG_RECORDER_NAME="default"
CREATED_NEW_CONFIG_RECORDER="false"

echo "Checking for existing configuration recorder..."
EXISTING_RECORDERS=$(aws configservice describe-configuration-recorders 2>/dev/
null || echo "")
```

```
if echo "$EXISTING_RECORDERS" | grep -q "name"; then
    echo "Configuration recorder already exists. Will update it."
    # Get the name of the existing recorder
    CONFIG_RECORDER_NAME=$(echo "$EXISTING_RECORDERS" | grep -o '"name": "[^"]*"'
 | head -1 | cut -d'"' -f4)
    echo "Using existing configuration recorder: $CONFIG_RECORDER_NAME"
else
    echo "No existing configuration recorder found. Will create a new one."
    CREATED_NEW_CONFIG_RECORDER="true"
fi

echo "Creating configuration recorder configuration..."
cat > configurationRecorder.json << EOF
{
  "name": "${CONFIG_RECORDER_NAME}",
  "roleARN": "${ROLE_ARN}",
  "recordingMode": {
    "recordingFrequency": "CONTINUOUS"
  }
}
EOF

echo "Creating recording group configuration..."
cat > recordingGroup.json << EOF
{
  "allSupported": true,
  "includeGlobalResourceTypes": true
}
EOF

echo "Setting up configuration recorder..."
RECORDER_RESULT=$(aws configservice put-configuration-recorder --configuration-
recorder file://configurationRecorder.json --recording-group file://
recordingGroup.json)
check_command "$RECORDER_RESULT"
echo "Configuration recorder set up"

if [ "$CREATED_NEW_CONFIG_RECORDER" = "true" ]; then
    aws configservice tag-resource --resource-arn "arn:aws:config:
${AWS_REGION}:${ACCOUNT_ID}:config-recorder/${CONFIG_RECORDER_NAME}" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-config-gs
    echo "Tags applied to configuration recorder"
fi
```

```
# Step 5: Check if delivery channel already exists
DELIVERY_CHANNEL_NAME="default"
CREATED_NEW_DELIVERY_CHANNEL="false"

echo "Checking for existing delivery channel..."
EXISTING_CHANNELS=$(aws configservice describe-delivery-channels 2>/dev/null ||
 echo "")
if echo "$EXISTING_CHANNELS" | grep -q "name"; then
    echo "Delivery channel already exists."
    # Get the name of the existing channel
    DELIVERY_CHANNEL_NAME=$(echo "$EXISTING_CHANNELS" | grep -o '"name": "[^"]*"'
 | head -1 | cut -d'"' -f4)
    echo "Using existing delivery channel: $DELIVERY_CHANNEL_NAME"

    # Update the existing delivery channel
    echo "Creating delivery channel configuration for update..."
    cat > deliveryChannel.json << EOF
{
  "name": "${DELIVERY_CHANNEL_NAME}",
  "s3BucketName": "${S3_BUCKET_NAME}",
  "snsTopicARN": "${SNS_TOPIC_ARN}",
  "configSnapshotDeliveryProperties": {
    "deliveryFrequency": "Six_Hours"
  }
}
EOF

    echo "Updating delivery channel..."
    CHANNEL_RESULT=$(aws configservice put-delivery-channel --delivery-channel
 file://deliveryChannel.json)
    check_command "$CHANNEL_RESULT"
    echo "Delivery channel updated"
else
    echo "No existing delivery channel found. Will create a new one."
    CREATED_NEW_DELIVERY_CHANNEL="true"

    echo "Creating delivery channel configuration..."
    cat > deliveryChannel.json << EOF
{
  "name": "${DELIVERY_CHANNEL_NAME}",
  "s3BucketName": "${S3_BUCKET_NAME}",
  "snsTopicARN": "${SNS_TOPIC_ARN}",
  "configSnapshotDeliveryProperties": {
    "deliveryFrequency": "Six_Hours"
```

```
  }
}
EOF

    echo "Creating delivery channel..."
    CHANNEL_RESULT=$(aws configservice put-delivery-channel --delivery-channel
 file://deliveryChannel.json)
    check_command "$CHANNEL_RESULT"
    echo "Delivery channel created"

    aws configservice tag-resource --resource-arn "arn:aws:config:
${AWS_REGION}:${ACCOUNT_ID}:delivery-channel/${DELIVERY_CHANNEL_NAME}" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-config-gs
    echo "Tags applied to delivery channel"
fi

# Step 6: Start the configuration recorder
echo "Checking configuration recorder status..."
RECORDER_STATUS=$(aws configservice describe-configuration-recorder-status 2>/
dev/null || echo "")
if echo "$RECORDER_STATUS" | grep -q '"recording": true'; then
    echo "Configuration recorder is already running."
else
    echo "Starting configuration recorder..."
    START_RESULT=$(aws configservice start-configuration-recorder --
configuration-recorder-name "$CONFIG_RECORDER_NAME")
    check_command "$START_RESULT"
    echo "Configuration recorder started"
fi

# Step 7: Verify the AWS Config setup
echo "Verifying delivery channel..."
VERIFY_CHANNEL=$(aws configservice describe-delivery-channels)
check_command "$VERIFY_CHANNEL"
echo "$VERIFY_CHANNEL"

echo "Verifying configuration recorder..."
VERIFY_RECORDER=$(aws configservice describe-configuration-recorders)
check_command "$VERIFY_RECORDER"
echo "$VERIFY_RECORDER"

echo "Verifying configuration recorder status..."
VERIFY_STATUS=$(aws configservice describe-configuration-recorder-status)
check_command "$VERIFY_STATUS"
```

```
echo "$VERIFY_STATUS"

# Display created resources
display_resources

# Ask if user wants to clean up resources
echo ""
echo "==========================================="
echo "CLEANUP CONFIRMATION"
echo "==========================================="
echo "Do you want to clean up all created resources? (y/n): "
CLEANUP_CHOICE='y'

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    echo "Cleaning up resources..."
    cleanup_resources
    echo "Cleanup completed."
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later."
fi

echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateBucket
  - CreateRole
  - CreateTopic
  - DeleteBucket
  - DeleteConfigurationRecorder
  - DeleteDeliveryChannel
  - DeleteRole
  - DeleteRolePolicy
  - DeleteTopic
  - DescribeConfigurationRecorderStatus
  - DescribeConfigurationRecorders
  - DescribeDeliveryChannels

- DetachRolePolicy
- GetCallerIdentity
- PutConfigurationRecorder
- PutDeliveryChannel
- PutPublicAccessBlock
- PutRolePolicy
- Rm
- StartConfigurationRecorder
- StopConfigurationRecorder

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with container registries

The following code example shows how to:

- Create a Docker image
- Create an Amazon ECR repository
- Delete resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon ECR Getting Started Script
# This script demonstrates the lifecycle of a Docker image in Amazon ECR
```

```
# Set up logging
LOG_FILE="ecr-tutorial.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "================================================="
echo "Amazon ECR Getting Started Tutorial"
echo "================================================="
echo "This script will:"
echo "1. Create a Docker image"
echo "2. Create an Amazon ECR repository"
echo "3. Authenticate to Amazon ECR"
echo "4. Push the image to Amazon ECR"
echo "5. Pull the image from Amazon ECR"
echo "6. Clean up resources (optional)"
echo "================================================="

# Check prerequisites
echo "Checking prerequisites..."

# Check if AWS CLI is installed
if ! command -v aws &> /dev/null; then
    echo "ERROR: AWS CLI is not installed. Please install it before running this
 script."
    echo "Visit https://docs.aws.amazon.com/cli/latest/userguide/install-
cliv2.html for installation instructions."
    exit 1
fi

# Check if AWS CLI is configured
if ! aws sts get-caller-identity &> /dev/null; then
    echo "ERROR: AWS CLI is not configured properly. Please run 'aws configure'
 to set up your credentials."
    exit 1
fi

# Check if Docker is installed
if ! command -v docker &> /dev/null; then
    echo "ERROR: Docker is not installed. Please install Docker before running
 this script."
    echo "Visit https://docs.docker.com/get-docker/ for installation
 instructions."
    exit 1
fi
```

```
# Check if Docker daemon is running
if ! docker info &> /dev/null; then
    echo "ERROR: Docker daemon is not running. Please start Docker and try
 again."
    exit 1
fi

echo "All prerequisites met."

# Initialize variables
REPO_URI=""
TIMEOUT_CMD="timeout 300"  # 5-minute timeout for long-running commands

# Function to handle errors
handle_error() {
    echo "ERROR: $1"
    echo "Check the log file for details: $LOG_FILE"

    echo "=================================================="
    echo "Resources created:"
    echo "- Docker image: hello-world (local)"
    if [ -n "$REPO_URI" ]; then
        echo "- ECR Repository: hello-repository"
        echo "- ECR Image: $REPO_URI:latest"
    fi
    echo "=================================================="

    echo "Attempting to clean up resources..."
    cleanup
    exit 1
}

# Function to clean up resources
cleanup() {
    echo "=================================================="
    echo "Cleaning up resources..."

    # Delete the image from ECR if it exists
    if [ -n "$REPO_URI" ]; then
        echo "Deleting image from ECR repository..."
        aws ecr batch-delete-image --repository-name hello-repository --image-ids
 imageTag=latest || echo "Failed to delete image, it may not exist or may have
 already been deleted."
```

```
    fi

    # Delete the ECR repository if it exists
    if [ -n "$REPO_URI" ]; then
        echo "Deleting ECR repository..."
        aws ecr delete-repository --repository-name hello-repository --force ||
 echo "Failed to delete repository, it may not exist or may have already been
 deleted."
    fi

    # Remove local Docker image
    echo "Removing local Docker image..."
    docker rmi hello-world:latest 2>/dev/null || echo "Failed to remove local
 image, it may not exist or may have already been deleted."
    if [ -n "$REPO_URI" ]; then
        docker rmi "$REPO_URI:latest" 2>/dev/null || echo "Failed to remove
 tagged image, it may not exist or may have already been deleted."
    fi

    echo "Cleanup completed."
    echo "================================================="
}

# Step 1: Create a Docker image
echo "Step 1: Creating a Docker image"

# Create Dockerfile
echo "Creating Dockerfile..."
cat > Dockerfile << 'EOF'
FROM public.ecr.aws/amazonlinux/amazonlinux:latest

# Install dependencies
RUN yum update -y && \
 yum install -y httpd

# Install apache and write hello world message
RUN echo 'Hello World!' > /var/www/html/index.html

# Configure apache
RUN echo 'mkdir -p /var/run/httpd' >> /root/run_apache.sh && \
 echo 'mkdir -p /var/lock/httpd' >> /root/run_apache.sh && \
 echo '/usr/sbin/httpd -D FOREGROUND' >> /root/run_apache.sh && \
 chmod 755 /root/run_apache.sh
```

```
EXPOSE 80

CMD /root/run_apache.sh
EOF

# Build Docker image
echo "Building Docker image..."
$TIMEOUT_CMD docker build -t hello-world . || handle_error "Failed to build
 Docker image or operation timed out after 5 minutes"

# Verify image was created
echo "Verifying Docker image..."
docker images --filter reference=hello-world || handle_error "Failed to list
 Docker images"

echo "Docker image created successfully."

# Step 2: Create an Amazon ECR repository
echo "Step 2: Creating an Amazon ECR repository"

# Get AWS account ID
echo "Getting AWS account ID..."
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
if [[ -z "$AWS_ACCOUNT_ID" || "$AWS_ACCOUNT_ID" == *"error"* ]]; then
    handle_error "Failed to get AWS account ID. Make sure your AWS credentials
 are configured correctly."
fi
echo "AWS Account ID: $AWS_ACCOUNT_ID"

# Get current region
AWS_REGION=$(aws configure get region)
if [[ -z "$AWS_REGION" ]]; then
    AWS_REGION="us-east-1"  # Default to us-east-1 if no region is configured
    echo "No AWS region configured, defaulting to $AWS_REGION"
else
    echo "Using AWS region: $AWS_REGION"
fi

# Create ECR repository
echo "Creating ECR repository..."
REPO_RESULT=$(aws ecr create-repository --repository-name hello-repository --
tags key=project,value=doc-smith key=tutorial,value=amazon-elastic-container-
registry-gs)
if [[ -z "$REPO_RESULT" || "$REPO_RESULT" == *"error"* ]]; then
```

```
    handle_error "Failed to create ECR repository"
fi

# Extract repository URI
REPO_URI="$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/hello-repository"
echo "Repository URI: $REPO_URI"

# Step 3: Authenticate to Amazon ECR
echo "Step 3: Authenticating to Amazon ECR"

echo "Getting ECR login password..."
aws ecr get-login-password --region "$AWS_REGION" | docker login --username
 AWS --password-stdin "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com" ||
 handle_error "Failed to authenticate to ECR"

echo "Successfully authenticated to ECR."

# Step 4: Push the image to Amazon ECR
echo "Step 4: Pushing the image to Amazon ECR"

# Tag the image
echo "Tagging Docker image..."
docker tag hello-world:latest "$REPO_URI:latest" || handle_error "Failed to tag
 Docker image"

# Push the image with timeout
echo "Pushing image to ECR..."
$TIMEOUT_CMD docker push "$REPO_URI:latest" || handle_error "Failed to push image
 to ECR or operation timed out after 5 minutes"

echo "Successfully pushed image to ECR."

# Step 5: Pull the image from Amazon ECR
echo "Step 5: Pulling the image from Amazon ECR"

# Remove local image to ensure we're pulling from ECR
echo "Removing local tagged image..."
docker rmi "$REPO_URI:latest" || echo "Warning: Failed to remove local tagged
 image"

# Pull the image with timeout
echo "Pulling image from ECR..."
$TIMEOUT_CMD docker pull "$REPO_URI:latest" || handle_error "Failed to pull image
 from ECR or operation timed out after 5 minutes"
```

```
echo "Successfully pulled image from ECR."

# List resources created
echo "================================================="
echo "Resources created:"
echo "- Docker image: hello-world (local)"
echo "- ECR Repository: hello-repository"
echo "- ECR Image: $REPO_URI:latest"
echo "================================================="

# Ask user if they want to clean up resources
echo ""
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    # Step 6: Delete the image from ECR
    echo "Step 6: Deleting the image from ECR"

    DELETE_IMAGE_RESULT=$(aws ecr batch-delete-image --repository-name hello-
repository --image-ids imageTag=latest)
    if [[ -z "$DELETE_IMAGE_RESULT" || "$DELETE_IMAGE_RESULT" == *"error"* ]];
 then
        echo "Warning: Failed to delete image from ECR"
    else
        echo "Successfully deleted image from ECR."
    fi

    # Step 7: Delete the ECR repository
    echo "Step 7: Deleting the ECR repository"

    DELETE_REPO_RESULT=$(aws ecr delete-repository --repository-name hello-
repository --force)
    if [[ -z "$DELETE_REPO_RESULT" || "$DELETE_REPO_RESULT" == *"error"* ]]; then
        echo "Warning: Failed to delete ECR repository"
    else
        echo "Successfully deleted ECR repository."
    fi

    # Remove local Docker images
```

```
    echo "Removing local Docker images..."
    docker rmi hello-world:latest 2>/dev/null || echo "Warning: Failed to remove
 local image"

    echo "All resources have been cleaned up."
else
    echo "Resources were not cleaned up. You can manually clean up later with:"
    echo "aws ecr batch-delete-image --repository-name hello-repository --image-
ids imageTag=latest"
    echo "aws ecr delete-repository --repository-name hello-repository --force"
    echo "docker rmi hello-world:latest"
    echo "docker rmi $REPO_URI:latest"
fi

echo "================================================="
echo "Tutorial completed!"
echo "Log file: $LOG_FILE"
echo "================================================="
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - BatchDeleteImage
  - CreateRepository
  - DeleteRepository
  - GetCallerIdentity
  - GetLoginPassword

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with internet of things messaging

The following code example shows how to:

- Create IoT resources
- Configure your device
- Run the sample application
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS IoT Core Getting Started Script
# This script creates AWS IoT resources, configures a device, and runs a sample
 application

# Set up logging
LOG_FILE="iot-core-setup.log"
echo "Starting AWS IoT Core setup at $(date)" > $LOG_FILE

# Function to log commands and their outputs
log_cmd() {
    echo "$(date): Running command: $1" >> $LOG_FILE
    eval "$1" 2>&1 | tee -a $LOG_FILE
    return ${PIPESTATUS[0]}
}

# Function to check for errors
check_error() {
    if [ $1 -ne 0 ]; then
        echo "ERROR: Command failed with exit code $1" | tee -a $LOG_FILE
        echo "Please check the log file $LOG_FILE for details" | tee -a $LOG_FILE
        cleanup_on_error
        exit $1
    fi
}

# Function to cleanup resources on error
cleanup_on_error() {
    echo "Error encountered. Attempting to clean up resources..." | tee -a
 $LOG_FILE
    echo "Resources created:" | tee -a $LOG_FILE
```

```
    if [ ! -z "$CERTIFICATE_ARN" ]; then
        echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
        if [ ! -z "$POLICY_NAME" ]; then
            log_cmd "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
        fi
        if [ ! -z "$THING_NAME" ]; then
            log_cmd "aws iot detach-thing-principal --thing-name $THING_NAME --
principal $CERTIFICATE_ARN"
        fi
        if [ ! -z "$CERTIFICATE_ID" ]; then
            log_cmd "aws iot update-certificate --certificate-id $CERTIFICATE_ID
 --new-status INACTIVE"
            log_cmd "aws iot delete-certificate --certificate-id $CERTIFICATE_ID"
        fi
    fi
    if [ ! -z "$THING_NAME" ]; then
        echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-thing --thing-name $THING_NAME"
    fi
    if [ ! -z "$POLICY_NAME" ]; then
        echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-policy --policy-name $POLICY_NAME"
    fi
    if [ ! -z "$SHARED_POLICY_NAME" ]; then
        echo "Shared Policy Name: $SHARED_POLICY_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-policy --policy-name $SHARED_POLICY_NAME"
    fi
}

# Generate unique identifiers
RANDOM_SUFFIX=$(openssl rand -hex 4)
THING_NAME="MyIoTThing-${RANDOM_SUFFIX}"
POLICY_NAME="MyIoTPolicy-${RANDOM_SUFFIX}"
SHARED_POLICY_NAME="SharedSubPolicy-${RANDOM_SUFFIX}"
CERTS_DIR="$HOME/certs"

echo "================================================" | tee -a $LOG_FILE
echo "AWS IoT Core Getting Started" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "This script will:" | tee -a $LOG_FILE
echo "1. Create AWS IoT resources (policy, thing, certificate)" | tee -a
 $LOG_FILE
echo "2. Configure your device" | tee -a $LOG_FILE
```

```
echo "3. Set up for running the sample application" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE
echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
echo "Certificates Directory: $CERTS_DIR" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE

# Get AWS account ID
echo "Getting AWS account ID..." | tee -a $LOG_FILE
ACCOUNT_ID=$(log_cmd "aws sts get-caller-identity --query Account --output text")
check_error $?

# Get AWS region
echo "Getting AWS region..." | tee -a $LOG_FILE
REGION=$(log_cmd "aws configure get region")
check_error $?
if [ -z "$REGION" ]; then
    echo "AWS region not configured. Please run 'aws configure' to set your
 region." | tee -a $LOG_FILE
    exit 1
fi

echo "Using AWS Account ID: $ACCOUNT_ID and Region: $REGION" | tee -a $LOG_FILE

# Step 1: Create AWS IoT Resources
echo "" | tee -a $LOG_FILE
echo "Step 1: Creating AWS IoT Resources..." | tee -a $LOG_FILE

# Create IoT policy
echo "Creating IoT policy document..." | tee -a $LOG_FILE
cat > iot-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:client/test-*"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topic/test/topic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/test/topic"
      ]
    }
  ]
}
EOF

echo "Creating IoT policy: $POLICY_NAME..." | tee -a $LOG_FILE
log_cmd "aws iot create-policy --policy-name $POLICY_NAME --policy-
document file://iot-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-core-gs"
check_error $?

# Create IoT thing
echo "Creating IoT thing: $THING_NAME..." | tee -a $LOG_FILE
log_cmd "aws iot create-thing --thing-name $THING_NAME"
check_error $?

THING_ARN=$(aws iot describe-thing --thing-name "$THING_NAME" --query 'thingArn'
 --output text)
aws iot tag-resource --resource-arn "$THING_ARN" --tags Key=project,Value=doc-
smith Key=tutorial,Value=aws-iot-core-gs

# Create directory for certificates
echo "Creating certificates directory..." | tee -a $LOG_FILE
log_cmd "mkdir -p $CERTS_DIR"
check_error $?
```

```
# Create keys and certificate
echo "Creating keys and certificate..." | tee -a $LOG_FILE
CERT_OUTPUT=$(log_cmd "aws iot create-keys-and-certificate --set-as-active
 --certificate-pem-outfile $CERTS_DIR/device.pem.crt --public-key-outfile
 $CERTS_DIR/public.pem.key --private-key-outfile $CERTS_DIR/private.pem.key")
check_error $?

# Extract certificate ARN and ID
CERTIFICATE_ARN=$(echo "$CERT_OUTPUT" | grep "certificateArn" | cut -d'"' -f4)
CERTIFICATE_ID=$(echo "$CERTIFICATE_ARN" | cut -d/ -f2)

if [ -z "$CERTIFICATE_ARN" ] || [ -z "$CERTIFICATE_ID" ]; then
    echo "Failed to extract certificate ARN or ID" | tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
echo "Certificate ID: $CERTIFICATE_ID" | tee -a $LOG_FILE

# Attach policy to certificate
echo "Attaching policy to certificate..." | tee -a $LOG_FILE
log_cmd "aws iot attach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
check_error $?

# Attach certificate to thing
echo "Attaching certificate to thing..." | tee -a $LOG_FILE
log_cmd "aws iot attach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN"
check_error $?

# Download Amazon Root CA certificate
echo "Downloading Amazon Root CA certificate..." | tee -a $LOG_FILE
log_cmd "curl -s -o $CERTS_DIR/Amazon-root-CA-1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem"
check_error $?

# Step 2: Configure Your Device
echo "" | tee -a $LOG_FILE
echo "Step 2: Configuring Your Device..." | tee -a $LOG_FILE

# Check if Git is installed
echo "Checking if Git is installed..." | tee -a $LOG_FILE
```

```
if ! command -v git &> /dev/null; then
    echo "Git is not installed. Please install Git and run this script again." |
 tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

# Check if Python is installed
echo "Checking if Python is installed..." | tee -a $LOG_FILE
if ! command -v python3 &> /dev/null; then
    echo "Python 3 is not installed. Please install Python 3 and run this script
 again." | tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

# Install AWS IoT Device SDK for Python
echo "Installing AWS IoT Device SDK for Python..." | tee -a $LOG_FILE
log_cmd "python3 -m pip install awsiotsdk"
check_error $?

# Clone the AWS IoT Device SDK for Python repository
echo "Cloning AWS IoT Device SDK for Python repository..." | tee -a $LOG_FILE
if [ ! -d "$HOME/aws-iot-device-sdk-python-v2" ]; then
    log_cmd "cd $HOME && git clone https://github.com/aws/aws-iot-device-sdk-
python-v2.git"
    check_error $?
else
    echo "AWS IoT Device SDK for Python repository already exists." | tee -a
 $LOG_FILE
fi

# Step 3: Get AWS IoT Endpoint
echo "" | tee -a $LOG_FILE
echo "Step 3: Getting AWS IoT Endpoint..." | tee -a $LOG_FILE

IOT_ENDPOINT=$(log_cmd "aws iot describe-endpoint --endpoint-type iot:Data-ATS --
query endpointAddress --output text")
check_error $?

echo "AWS IoT Endpoint: $IOT_ENDPOINT" | tee -a $LOG_FILE

# Create a shared subscription policy (optional)
echo "" | tee -a $LOG_FILE
```

```
echo "Creating shared subscription policy (optional)..." | tee -a $LOG_FILE

cat > shared-sub-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:client/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topic/test/topic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/test/topic",
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/\$share/*/test/topic"
      ]
    }
  ]
}
EOF

log_cmd "aws iot create-policy --policy-name $SHARED_POLICY_NAME --policy-
document file://shared-sub-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-core-gs"
check_error $?
```

```
log_cmd "aws iot attach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN"
check_error $?

# Summary of created resources
echo "" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Setup Complete! Resources Created:" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
echo "Shared Subscription Policy Name: $SHARED_POLICY_NAME" | tee -a $LOG_FILE
echo "Certificate ID: $CERTIFICATE_ID" | tee -a $LOG_FILE
echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
echo "Certificate Files Location: $CERTS_DIR" | tee -a $LOG_FILE
echo "AWS IoT Endpoint: $IOT_ENDPOINT" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE

# Instructions for running the sample application
echo "" | tee -a $LOG_FILE
echo "To run the sample application, execute:" | tee -a $LOG_FILE
echo "cd $HOME/aws-iot-device-sdk-python-v2/samples" | tee -a $LOG_FILE
echo "python3 pubsub.py \\" | tee -a $LOG_FILE
echo "  --endpoint $IOT_ENDPOINT \\" | tee -a $LOG_FILE
echo "  --ca_file $CERTS_DIR/Amazon-root-CA-1.pem \\" | tee -a $LOG_FILE
echo "  --cert $CERTS_DIR/device.pem.crt \\" | tee -a $LOG_FILE
echo "  --key $CERTS_DIR/private.pem.key" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE
echo "To run the shared subscription example, execute:" | tee -a $LOG_FILE
echo "cd $HOME/aws-iot-device-sdk-python-v2/samples" | tee -a $LOG_FILE
echo "python3 mqtt5_shared_subscription.py \\" | tee -a $LOG_FILE
echo "  --endpoint $IOT_ENDPOINT \\" | tee -a $LOG_FILE
echo "  --ca_file $CERTS_DIR/Amazon-root-CA-1.pem \\" | tee -a $LOG_FILE
echo "  --cert $CERTS_DIR/device.pem.crt \\" | tee -a $LOG_FILE
echo "  --key $CERTS_DIR/private.pem.key \\" | tee -a $LOG_FILE
echo "  --group_identifier consumer" | tee -a $LOG_FILE

# Ask if user wants to clean up resources
echo "" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "CLEANUP CONFIRMATION" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Do you want to clean up all created resources? (y/n): " | tee -a $LOG_FILE
read -r CLEANUP_CHOICE
```

```
if [[ $CLEANUP_CHOICE =~ ^[Yy]$ ]]; then
    echo "Cleaning up resources..." | tee -a $LOG_FILE

    # Detach policies from certificate
    echo "Detaching policies from certificate..." | tee -a $LOG_FILE
    log_cmd "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
    log_cmd "aws iot detach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN"

    # Detach certificate from thing
    echo "Detaching certificate from thing..." | tee -a $LOG_FILE
    log_cmd "aws iot detach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN"

    # Update certificate status to INACTIVE
    echo "Setting certificate to inactive..." | tee -a $LOG_FILE
    log_cmd "aws iot update-certificate --certificate-id $CERTIFICATE_ID --new-
status INACTIVE"

    # Delete certificate
    echo "Deleting certificate..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-certificate --certificate-id $CERTIFICATE_ID"

    # Delete thing
    echo "Deleting thing..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-thing --thing-name $THING_NAME"

    # Delete policies
    echo "Deleting policies..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-policy --policy-name $POLICY_NAME"
    log_cmd "aws iot delete-policy --policy-name $SHARED_POLICY_NAME"

    echo "Cleanup complete!" | tee -a $LOG_FILE
else
    echo "Resources were not cleaned up. You can manually clean them up later." |
 tee -a $LOG_FILE
    echo "To clean up resources, run the following commands:" | tee -a $LOG_FILE
    echo "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
    echo "aws iot detach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
```

```
    echo "aws iot detach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
    echo "aws iot update-certificate --certificate-id $CERTIFICATE_ID --new-
status INACTIVE" | tee -a $LOG_FILE
    echo "aws iot delete-certificate --certificate-id $CERTIFICATE_ID" | tee -a
 $LOG_FILE
    echo "aws iot delete-thing --thing-name $THING_NAME" | tee -a $LOG_FILE
    echo "aws iot delete-policy --policy-name $POLICY_NAME" | tee -a $LOG_FILE
    echo "aws iot delete-policy --policy-name $SHARED_POLICY_NAME" | tee -a
 $LOG_FILE
fi

echo "" | tee -a $LOG_FILE
echo "Script execution completed. See $LOG_FILE for details." | tee -a $LOG_FILE
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachPolicy
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreatePolicy
  - CreateThing
  - DeleteCertificate
  - DeletePolicy
  - DeleteThing
  - DescribeEndpoint
  - DetachPolicy
  - DetachThingPrincipal
  - GetCallerIdentity
  - UpdateCertificate

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with machine learning feature stores

The following code example shows how to:

- Set up IAM permissions
- Create a SageMaker execution role
- Create feature groups
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon SageMaker Feature Store Tutorial Script - Version 3
# This script demonstrates how to use Amazon SageMaker Feature Store with AWS CLI

# Setup logging
LOG_FILE="sagemaker-featurestore-tutorial.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting SageMaker Feature Store tutorial script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"
echo ""

# Track created resources for cleanup
CREATED_RESOURCES=()

# Function to handle errors
handle_error() {
    echo "ERROR: $1"
    echo "Attempting to clean up resources..."
    cleanup_resources
```

```
    exit 1
}

# Function to check command status
check_status() {
    if echo "$1" | grep -i "error" > /dev/null; then
        handle_error "$1"
    fi
}

# Function to wait for feature group to be created
wait_for_feature_group() {
    local feature_group_name=$1
    local status="Creating"

    echo "Waiting for feature group ${feature_group_name} to be created..."

    while [ "$status" = "Creating" ]; do
        sleep 5
        status=$(aws sagemaker describe-feature-group \
            --feature-group-name "${feature_group_name}" \
            --query 'FeatureGroupStatus' \
            --output text)
        echo "Current status: ${status}"

        if [ "$status" = "Failed" ]; then
            handle_error "Feature group ${feature_group_name} creation failed"
        fi
    done

    echo "Feature group ${feature_group_name} is now ${status}"
}

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources..."

    # Clean up in reverse order
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        resource="${CREATED_RESOURCES[$i]}"
        resource_type=$(echo "$resource" | cut -d: -f1)
        resource_name=$(echo "$resource" | cut -d: -f2)

        echo "Deleting $resource_type: $resource_name"
```

```
        case "$resource_type" in
            "FeatureGroup")
                aws sagemaker delete-feature-group --feature-group-name
 "$resource_name"
                ;;
            "S3Bucket")
                echo "Emptying S3 bucket: $resource_name"
                aws s3 rm "s3://$resource_name" --recursive 2>/dev/null
                echo "Deleting S3 bucket: $resource_name"
                aws s3api delete-bucket --bucket "$resource_name" 2>/dev/null
                ;;
            "IAMRole")
                echo "Detaching policies from role: $resource_name"
                aws iam detach-role-policy --role-name "$resource_name" --policy-
arn "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess" 2>/dev/null
                aws iam detach-role-policy --role-name "$resource_name" --policy-
arn "arn:aws:iam::aws:policy/AmazonS3FullAccess" 2>/dev/null
                echo "Deleting IAM role: $resource_name"
                aws iam delete-role --role-name "$resource_name" 2>/dev/null
                ;;
            *)
                echo "Unknown resource type: $resource_type"
                ;;
        esac
    done
}

# Function to create SageMaker execution role
create_sagemaker_role() {
    local role_name="SageMakerFeatureStoreRole-$(openssl rand -hex 4)"

    echo "Creating SageMaker execution role: $role_name" >&2

    # Create trust policy document
    local trust_policy='{
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "sagemaker.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
```

```
            }
        ]
    }'

    # Create the role
    local role_result=$(aws iam create-role \
        --role-name "$role_name" \
        --assume-role-policy-document "$trust_policy" \
        --description "SageMaker execution role for Feature Store tutorial" 2>&1)

    if echo "$role_result" | grep -i "error" > /dev/null; then
        handle_error "Failed to create IAM role: $role_result"
    fi

    echo "Role created successfully" >&2
    CREATED_RESOURCES+=("IAMRole:$role_name")

    # Tag the role
    echo "Tagging IAM role..." >&2
    aws iam tag-role --role-name "$role_name" --tags Key=project,Value=doc-smith
Key=tutorial,Value=sagemaker-featurestore 2>&1

    # Attach necessary policies
    echo "Attaching policies to role..." >&2

    # SageMaker execution policy
    local policy1_result=$(aws iam attach-role-policy \
        --role-name "$role_name" \
        --policy-arn "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess" 2>&1)

    if echo "$policy1_result" | grep -i "error" > /dev/null; then
        handle_error "Failed to attach SageMaker policy: $policy1_result"
    fi

    # S3 access policy
    local policy2_result=$(aws iam attach-role-policy \
        --role-name "$role_name" \
        --policy-arn "arn:aws:iam::aws:policy/AmazonS3FullAccess" 2>&1)

    if echo "$policy2_result" | grep -i "error" > /dev/null; then
        handle_error "Failed to attach S3 policy: $policy2_result"
    fi

    # Get account ID for role ARN
```

```
    local account_id=$(aws sts get-caller-identity --query Account --output text)
    local role_arn="arn:aws:iam::${account_id}:role/${role_name}"

    echo "Role ARN: $role_arn" >&2
    echo "Waiting 10 seconds for role to propagate..." >&2
    sleep 10

    # Return only the role ARN to stdout
    echo "$role_arn"
}

# Handle SageMaker execution role
ROLE_ARN=""

if [ -z "$1" ]; then
    echo "Creating SageMaker execution role automatically..."
    ROLE_ARN=$(create_sagemaker_role)
    if [ -z "$ROLE_ARN" ]; then
        handle_error "Failed to create SageMaker execution role"
    fi
else
    ROLE_ARN="$1"

    # Validate the role ARN
    ROLE_NAME=$(echo "$ROLE_ARN" | sed 's/.*role\///')
    ROLE_CHECK=$(aws iam get-role --role-name "$ROLE_NAME" 2>&1)
    if echo "$ROLE_CHECK" | grep -i "error" > /dev/null; then
        echo "Creating a new role automatically..."
        ROLE_ARN=$(create_sagemaker_role)
        if [ -z "$ROLE_ARN" ]; then
            handle_error "Failed to create SageMaker execution role"
        fi
    fi
fi

# Handle cleanup option
AUTO_CLEANUP=""
if [ -n "$2" ]; then
    AUTO_CLEANUP="$2"
fi

# Generate a random identifier for resource names
RANDOM_ID=$(openssl rand -hex 4)
echo "Using random identifier: $RANDOM_ID"
```

```
# Set variables
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
check_status "$ACCOUNT_ID"
echo "Account ID: $ACCOUNT_ID"

# Get current region
REGION=$(aws configure get region)
if [ -z "$REGION" ]; then
    REGION="us-east-1"
    echo "No default region configured, using: $REGION"
else
    echo "Using region: $REGION"
fi
# Check for shared prereq bucket
PREREQ_BUCKET=$(aws cloudformation describe-stacks --stack-name tutorial-prereqs-
bucket \
    --query 'Stacks[0].Outputs[?OutputKey==`BucketName`].OutputValue' --output
 text 2>/dev/null)
if [ -n "$PREREQ_BUCKET" ] && [ "$PREREQ_BUCKET" != "None" ]; then
    S3_BUCKET_NAME="$PREREQ_BUCKET"
    BUCKET_IS_SHARED=true
    echo "Using shared bucket: $S3_BUCKET_NAME"
else
    BUCKET_IS_SHARED=false
    S3_BUCKET_NAME="sagemaker-featurestore-${RANDOM_ID}-${ACCOUNT_ID}"
fi
PREFIX="featurestore-tutorial"
CURRENT_TIME=$(date +%s)

echo "Creating S3 bucket: $S3_BUCKET_NAME"
# Create bucket in current region (skip if using shared bucket)
if [ "$BUCKET_IS_SHARED" = "false" ]; then
if [ "$REGION" = "us-east-1" ]; then
    BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME" \
        --region "$REGION" 2>&1)
else
    BUCKET_RESULT=$(aws s3api create-bucket --bucket "$S3_BUCKET_NAME" \
        --region "$REGION" \
        --create-bucket-configuration LocationConstraint="$REGION" 2>&1)
fi

if echo "$BUCKET_RESULT" | grep -i "error" > /dev/null; then
    echo "Failed to create S3 bucket: $BUCKET_RESULT"
```

```
    exit 1
fi

echo "$BUCKET_RESULT"
CREATED_RESOURCES+=("S3Bucket:$S3_BUCKET_NAME")

# Tag the S3 bucket
echo "Tagging S3 bucket: $S3_BUCKET_NAME"
aws s3api put-bucket-tagging --bucket "$S3_BUCKET_NAME" --tagging
 'TagSet=[{Key=project,Value=doc-smith},{Key=tutorial,Value=sagemaker-
featurestore}]' 2>&1

# Block public access to the bucket
BLOCK_RESULT=$(aws s3api put-public-access-block \
    --bucket "$S3_BUCKET_NAME" \
    --public-access-block-configuration
 "BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets
 2>&1)

if echo "$BLOCK_RESULT" | grep -i "error" > /dev/null; then
    echo "Failed to block public access to S3 bucket: $BLOCK_RESULT"
    cleanup_resources
    exit 1
fi
else
    echo "Using shared bucket (skipping creation)"
fi

# Create feature groups
echo "Creating feature groups..."

# Create customers feature group
CUSTOMERS_FEATURE_GROUP_NAME="customers-feature-group-${RANDOM_ID}"
echo "Creating customers feature group: $CUSTOMERS_FEATURE_GROUP_NAME"

CUSTOMERS_RESPONSE=$(aws sagemaker create-feature-group \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record-identifier-feature-name "customer_id" \
    --event-time-feature-name "EventTime" \
    --feature-definitions '[
        {"FeatureName": "customer_id", "FeatureType": "Integral"},
        {"FeatureName": "name", "FeatureType": "String"},
        {"FeatureName": "age", "FeatureType": "Integral"},
        {"FeatureName": "address", "FeatureType": "String"},
```

```
        {"FeatureName": "membership_type", "FeatureType": "String"},
        {"FeatureName": "EventTime", "FeatureType": "Fractional"}
    ]' \
    --online-store-config '{"EnableOnlineStore": true}' \
    --offline-store-config '{
        "S3StorageConfig": {
            "S3Uri": "s3://'${S3_BUCKET_NAME}'/'${PREFIX}'"
        },
        "DisableGlueTableCreation": false
    }' \
    --role-arn "$ROLE_ARN" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=sagemaker-featurestore
 2>&1)

if echo "$CUSTOMERS_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to create customers feature group: $CUSTOMERS_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMERS_RESPONSE"
CREATED_RESOURCES+=("FeatureGroup:$CUSTOMERS_FEATURE_GROUP_NAME")

# Create orders feature group
ORDERS_FEATURE_GROUP_NAME="orders-feature-group-${RANDOM_ID}"
echo "Creating orders feature group: $ORDERS_FEATURE_GROUP_NAME"

ORDERS_RESPONSE=$(aws sagemaker create-feature-group \
    --feature-group-name "$ORDERS_FEATURE_GROUP_NAME" \
    --record-identifier-feature-name "customer_id" \
    --event-time-feature-name "EventTime" \
    --feature-definitions '[
        {"FeatureName": "customer_id", "FeatureType": "Integral"},
        {"FeatureName": "order_id", "FeatureType": "String"},
        {"FeatureName": "order_date", "FeatureType": "String"},
        {"FeatureName": "product", "FeatureType": "String"},
        {"FeatureName": "quantity", "FeatureType": "Integral"},
        {"FeatureName": "amount", "FeatureType": "Fractional"},
        {"FeatureName": "EventTime", "FeatureType": "Fractional"}
    ]' \
    --online-store-config '{"EnableOnlineStore": true}' \
    --offline-store-config '{
        "S3StorageConfig": {
            "S3Uri": "s3://'${S3_BUCKET_NAME}'/'${PREFIX}'"
```

```
        },
        "DisableGlueTableCreation": false
    }' \
    --role-arn "$ROLE_ARN" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=sagemaker-featurestore
 2>&1)

if echo "$ORDERS_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to create orders feature group: $ORDERS_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$ORDERS_RESPONSE"
CREATED_RESOURCES+=("FeatureGroup:$ORDERS_FEATURE_GROUP_NAME")

# Wait for feature groups to be created
wait_for_feature_group "$CUSTOMERS_FEATURE_GROUP_NAME"
wait_for_feature_group "$ORDERS_FEATURE_GROUP_NAME"

# Ingest data into feature groups
echo "Ingesting data into feature groups..."

# Ingest customer data
echo "Ingesting customer data..."
CUSTOMER1_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record '[
        {"FeatureName": "customer_id", "ValueAsString": "573291"},
        {"FeatureName": "name", "ValueAsString": "John Doe"},
        {"FeatureName": "age", "ValueAsString": "35"},
        {"FeatureName": "address", "ValueAsString": "123 Main St"},
        {"FeatureName": "membership_type", "ValueAsString": "premium"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$CUSTOMER1_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest customer 1 data: $CUSTOMER1_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMER1_RESPONSE"
```

```
CUSTOMER2_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record '[
        {"FeatureName": "customer_id", "ValueAsString": "109382"},
        {"FeatureName": "name", "ValueAsString": "Jane Smith"},
        {"FeatureName": "age", "ValueAsString": "28"},
        {"FeatureName": "address", "ValueAsString": "456 Oak Ave"},
        {"FeatureName": "membership_type", "ValueAsString": "standard"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$CUSTOMER2_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest customer 2 data: $CUSTOMER2_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMER2_RESPONSE"

# Ingest order data
echo "Ingesting order data..."
ORDER1_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
    --feature-group-name "$ORDERS_FEATURE_GROUP_NAME" \
    --record '[
        {"FeatureName": "customer_id", "ValueAsString": "573291"},
        {"FeatureName": "order_id", "ValueAsString": "ORD-001"},
        {"FeatureName": "order_date", "ValueAsString": "2023-01-15"},
        {"FeatureName": "product", "ValueAsString": "Laptop"},
        {"FeatureName": "quantity", "ValueAsString": "1"},
        {"FeatureName": "amount", "ValueAsString": "1299.99"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$ORDER1_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest order 1 data: $ORDER1_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$ORDER1_RESPONSE"

ORDER2_RESPONSE=$(aws sagemaker-featurestore-runtime put-record \
    --feature-group-name "$ORDERS_FEATURE_GROUP_NAME" \
    --record '[
```

```
        {"FeatureName": "customer_id", "ValueAsString": "109382"},
        {"FeatureName": "order_id", "ValueAsString": "ORD-002"},
        {"FeatureName": "order_date", "ValueAsString": "2023-01-20"},
        {"FeatureName": "product", "ValueAsString": "Smartphone"},
        {"FeatureName": "quantity", "ValueAsString": "1"},
        {"FeatureName": "amount", "ValueAsString": "899.99"},
        {"FeatureName": "EventTime", "ValueAsString": "'${CURRENT_TIME}'"}
    ]' 2>&1)

if echo "$ORDER2_RESPONSE" | grep -i "error" > /dev/null; then
    echo "Failed to ingest order 2 data: $ORDER2_RESPONSE"
    cleanup_resources
    exit 1
fi

echo "$ORDER2_RESPONSE"

# Retrieve records from feature groups
echo "Retrieving records from feature groups..."

# Get a single customer record
echo "Getting customer record with ID 573291:"
CUSTOMER_RECORD=$(aws sagemaker-featurestore-runtime get-record \
    --feature-group-name "$CUSTOMERS_FEATURE_GROUP_NAME" \
    --record-identifier-value-as-string "573291" 2>&1)

if echo "$CUSTOMER_RECORD" | grep -i "error" > /dev/null; then
    echo "Failed to get customer record: $CUSTOMER_RECORD"
    cleanup_resources
    exit 1
fi

echo "$CUSTOMER_RECORD"

# Get multiple records using batch-get-record
echo "Getting multiple records using batch-get-record:"
BATCH_RECORDS=$(aws sagemaker-featurestore-runtime batch-get-record \
    --identifiers '[
        {
            "FeatureGroupName": "'${CUSTOMERS_FEATURE_GROUP_NAME}'",
            "RecordIdentifiersValueAsString": ["573291", "109382"]
        },
        {
            "FeatureGroupName": "'${ORDERS_FEATURE_GROUP_NAME}'",
```

```
            "RecordIdentifiersValueAsString": ["573291", "109382"]
        }
    ]' 2>&1)

if echo "$BATCH_RECORDS" | grep -i "error" > /dev/null && ! echo "$BATCH_RECORDS"
 | grep -i "Records" > /dev/null; then
    echo "Failed to get batch records: $BATCH_RECORDS"
    cleanup_resources
    exit 1
fi

echo "$BATCH_RECORDS"

# List feature groups
echo "Listing feature groups:"
FEATURE_GROUPS=$(aws sagemaker list-feature-groups 2>&1)

if echo "$FEATURE_GROUPS" | grep -i "error" > /dev/null; then
    echo "Failed to list feature groups: $FEATURE_GROUPS"
    cleanup_resources
    exit 1
fi

echo "$FEATURE_GROUPS"

# Display summary of created resources
echo ""
echo "==========================================="
echo "TUTORIAL COMPLETED SUCCESSFULLY!"
echo "==========================================="
echo "Resources created:"
echo "- S3 Bucket: $S3_BUCKET_NAME"
echo "- Customers Feature Group: $CUSTOMERS_FEATURE_GROUP_NAME"
echo "- Orders Feature Group: $ORDERS_FEATURE_GROUP_NAME"
if [[ " ${CREATED_RESOURCES[@]} " =~ " IAMRole:" ]]; then
    echo "- IAM Role: $(echo "${CREATED_RESOURCES[@]}" | grep -o 'IAMRole:
[^[:space:]]*' | cut -d: -f2)"
fi
echo ""
echo "You can now:"
echo "1. View your feature groups in the SageMaker console"
echo "2. Query the offline store using Amazon Athena"
echo "3. Use the feature groups in your ML workflows"
echo "==========================================="
```

```
echo ""

# Handle cleanup
if [ "$AUTO_CLEANUP" = "y" ]; then
    echo "Auto-cleanup enabled. Starting cleanup..."
    cleanup_resources
    echo "Cleanup completed."
elif [ "$AUTO_CLEANUP" = "n" ]; then
    echo "Auto-cleanup disabled. Resources will remain in your account."
    echo "To clean up later, run this script again with cleanup option 'y'"
else
    echo "==========================================="
    echo "CLEANUP CONFIRMATION"
    echo "==========================================="
    echo "Do you want to clean up all created resources? (y/n): "
    read -r CLEANUP_CHOICE

    if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
        echo "Starting cleanup..."
        cleanup_resources
        echo "Cleanup completed."
    else
        echo "Skipping cleanup. Resources will remain in your account."
        echo "To clean up later, delete the following resources:"
        echo "- Feature Groups: $CUSTOMERS_FEATURE_GROUP_NAME,
 $ORDERS_FEATURE_GROUP_NAME"
        echo "- S3 Bucket: $S3_BUCKET_NAME"
        if [[ " ${CREATED_RESOURCES[@]} " =~ " IAMRole:" ]]; then
            echo "- IAM Role: $(echo "${CREATED_RESOURCES[@]}" | grep -o
 'IAMRole:[^[:space:]]*' | cut -d: -f2)"
        fi
        echo ""
        echo "Estimated ongoing cost: ~$0.01 per month for online store"
    fi
fi

echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateBucket
  - CreateFeatureGroup

- CreateRole
- DeleteBucket
- DeleteFeatureGroup
- DeleteRole
- DescribeFeatureGroup
- DetachRolePolicy
- GetCallerIdentity
- GetRole
- ListFeatureGroups
- PutPublicAccessBlock
- Rm

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with managed kubernetes clusters

The following code example shows how to:

- Create a VPC for your EKS cluster
- Create IAM roles for your EKS cluster
- Create your EKS cluster
- Configure kubectl to communicate with your cluster
- Create a managed node group
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon EKS Cluster Creation Script (v2)
# This script creates an Amazon EKS cluster with a managed node group using the
 AWS CLI

# Set up logging
LOG_FILE="eks-cluster-creation-v2.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon EKS cluster creation script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Error handling function
handle_error() {
    echo "ERROR: $1"
    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Function to check command success
check_command() {
    if [ $? -ne 0 ] || echo "$1" | grep -i "error" > /dev/null; then
        handle_error "$1"
    fi
}

# Function to check if kubectl is installed
check_kubectl() {
    if ! command -v kubectl &> /dev/null; then
        echo "WARNING: kubectl is not installed or not in your PATH."
```

```
        echo ""
        echo "To install kubectl, follow these instructions based on your
 operating system:"
        echo ""
        echo "For Linux:"
        echo "  1. Download the latest release:"
        echo "     curl -LO \"https://dl.k8s.io/release/\$(curl -L -s https://
dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl\""
        echo ""
        echo "  2. Make the kubectl binary executable:"
        echo "     chmod +x ./kubectl"
        echo ""
        echo "  3. Move the binary to your PATH:"
        echo "     sudo mv ./kubectl /usr/local/bin/kubectl"
        echo ""
        echo "For macOS:"
        echo "  1. Using Homebrew:"
        echo "     brew install kubectl"
        echo "     or"
        echo "  2. Using curl:"
        echo "     curl -LO \"https://dl.k8s.io/release/\$(curl -L -s https://
dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl\""
        echo "     chmod +x ./kubectl"
        echo "     sudo mv ./kubectl /usr/local/bin/kubectl"
        echo ""
        echo "For Windows:"
        echo "  1. Using curl:"
        echo "     curl -LO \"https://dl.k8s.io/release/v1.28.0/bin/windows/
amd64/kubectl.exe\""
        echo "     Add the binary to your PATH"
        echo "     or"
        echo "  2. Using Chocolatey:"
        echo "     choco install kubernetes-cli"
        echo ""
        echo "After installation, verify with: kubectl version --client"
        echo ""
        return 1
    fi
    return 0
}

# Generate a random identifier for resource names
RANDOM_ID=$(LC_ALL=C tr -dc 'a-z0-9' < /dev/urandom | fold -w 6 | head -n 1)
STACK_NAME="eks-vpc-stack-${RANDOM_ID}"
```

```
CLUSTER_NAME="eks-cluster-${RANDOM_ID}"
NODEGROUP_NAME="eks-nodegroup-${RANDOM_ID}"
CLUSTER_ROLE_NAME="EKSClusterRole-${RANDOM_ID}"
NODE_ROLE_NAME="EKSNodeRole-${RANDOM_ID}"

echo "Using the following resource names:"
echo "- VPC Stack: $STACK_NAME"
echo "- EKS Cluster: $CLUSTER_NAME"
echo "- Node Group: $NODEGROUP_NAME"
echo "- Cluster IAM Role: $CLUSTER_ROLE_NAME"
echo "- Node IAM Role: $NODE_ROLE_NAME"

# Array to track created resources for cleanup
declare -a CREATED_RESOURCES

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources in reverse order..."

    # Check if node group exists and delete it
    if aws eks list-nodegroups --cluster-name "$CLUSTER_NAME" --query
 "nodegroups[?contains(@,'$NODEGROUP_NAME')]" --output text 2>/dev/null | grep -q
 "$NODEGROUP_NAME"; then
        echo "Deleting node group: $NODEGROUP_NAME"
        aws eks delete-nodegroup --cluster-name "$CLUSTER_NAME" --nodegroup-name
 "$NODEGROUP_NAME"
        echo "Waiting for node group deletion to complete..."
        aws eks wait nodegroup-deleted --cluster-name "$CLUSTER_NAME" --
nodegroup-name "$NODEGROUP_NAME"
        echo "Node group deleted successfully."
    fi

    # Check if cluster exists and delete it
    if aws eks describe-cluster --name "$CLUSTER_NAME" 2>/dev/null; then
        echo "Deleting cluster: $CLUSTER_NAME"
        aws eks delete-cluster --name "$CLUSTER_NAME"
        echo "Waiting for cluster deletion to complete (this may take several
 minutes)..."
        aws eks wait cluster-deleted --name "$CLUSTER_NAME"
        echo "Cluster deleted successfully."
    fi

    # Check if CloudFormation stack exists and delete it
```

```
    if aws cloudformation describe-stacks --stack-name "$STACK_NAME" 2>/dev/null;
 then
        echo "Deleting CloudFormation stack: $STACK_NAME"
        aws cloudformation delete-stack --stack-name "$STACK_NAME"
        echo "Waiting for CloudFormation stack deletion to complete..."
        aws cloudformation wait stack-delete-complete --stack-name "$STACK_NAME"
        echo "CloudFormation stack deleted successfully."
    fi

    # Clean up IAM roles
    if aws iam get-role --role-name "$NODE_ROLE_NAME" 2>/dev/null; then
        echo "Detaching policies from node role: $NODE_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSWorkerNodePolicy --role-name "$NODE_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly --role-name "$NODE_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKS_CNI_Policy --role-name "$NODE_ROLE_NAME"
        echo "Deleting node role: $NODE_ROLE_NAME"
        aws iam delete-role --role-name "$NODE_ROLE_NAME"
        echo "Node role deleted successfully."
    fi

    if aws iam get-role --role-name "$CLUSTER_ROLE_NAME" 2>/dev/null; then
        echo "Detaching policies from cluster role: $CLUSTER_ROLE_NAME"
        aws iam detach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name "$CLUSTER_ROLE_NAME"
        echo "Deleting cluster role: $CLUSTER_ROLE_NAME"
        aws iam delete-role --role-name "$CLUSTER_ROLE_NAME"
        echo "Cluster role deleted successfully."
    fi

    echo "Cleanup complete."
}

# Trap to ensure cleanup on script exit
trap 'echo "Script interrupted. Cleaning up resources..."; cleanup_resources;
 exit 1' SIGINT SIGTERM

# Verify AWS CLI configuration
echo "Verifying AWS CLI configuration..."
AWS_ACCOUNT_INFO=$(aws sts get-caller-identity)
check_command "$AWS_ACCOUNT_INFO"
echo "AWS CLI is properly configured."
```

```
# Step 1: Create VPC using CloudFormation
echo "Step 1: Creating VPC with CloudFormation..."
echo "Creating CloudFormation stack: $STACK_NAME"

# Create the CloudFormation stack
CF_CREATE_OUTPUT=$(aws cloudformation create-stack \
  --stack-name "$STACK_NAME" \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=eks-gs)
check_command "$CF_CREATE_OUTPUT"
CREATED_RESOURCES+=("CloudFormation Stack: $STACK_NAME")

echo "Waiting for CloudFormation stack to complete (this may take a few
 minutes)..."
aws cloudformation wait stack-create-complete --stack-name "$STACK_NAME"
if [ $? -ne 0 ]; then
    handle_error "CloudFormation stack creation failed"
fi
echo "CloudFormation stack created successfully."

# Step 2: Create IAM roles for EKS
echo "Step 2: Creating IAM roles for EKS..."

# Create cluster role trust policy
echo "Creating cluster role trust policy..."
cat > eks-cluster-role-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create cluster role
echo "Creating cluster IAM role: $CLUSTER_ROLE_NAME"
```

```
CLUSTER_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$CLUSTER_ROLE_NAME" \
  --assume-role-policy-document file://"eks-cluster-role-trust-policy.json")
check_command "$CLUSTER_ROLE_OUTPUT"
aws iam tag-role --role-name "$CLUSTER_ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=eks-gs
CREATED_RESOURCES+=("IAM Role: $CLUSTER_ROLE_NAME")

# Attach policy to cluster role
echo "Attaching EKS cluster policy to role..."
ATTACH_CLUSTER_POLICY_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name "$CLUSTER_ROLE_NAME")
check_command "$ATTACH_CLUSTER_POLICY_OUTPUT"

# Create node role trust policy
echo "Creating node role trust policy..."
cat > node-role-trust-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create node role
echo "Creating node IAM role: $NODE_ROLE_NAME"
NODE_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$NODE_ROLE_NAME" \
  --assume-role-policy-document file://"node-role-trust-policy.json")
check_command "$NODE_ROLE_OUTPUT"
aws iam tag-role --role-name "$NODE_ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=eks-gs
CREATED_RESOURCES+=("IAM Role: $NODE_ROLE_NAME")

# Attach policies to node role
echo "Attaching EKS node policies to role..."
```

```
ATTACH_NODE_POLICY1_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name "$NODE_ROLE_NAME")
check_command "$ATTACH_NODE_POLICY1_OUTPUT"

ATTACH_NODE_POLICY2_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name "$NODE_ROLE_NAME")
check_command "$ATTACH_NODE_POLICY2_OUTPUT"

ATTACH_NODE_POLICY3_OUTPUT=$(aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name "$NODE_ROLE_NAME")
check_command "$ATTACH_NODE_POLICY3_OUTPUT"

# Step 3: Get VPC and subnet information
echo "Step 3: Getting VPC and subnet information..."

VPC_ID=$(aws cloudformation describe-stacks \
  --stack-name "$STACK_NAME" \
  --query "Stacks[0].Outputs[?OutputKey=='VpcId'].OutputValue" \
  --output text)
if [ -z "$VPC_ID" ]; then
    handle_error "Failed to get VPC ID from CloudFormation stack"
fi
echo "VPC ID: $VPC_ID"

SUBNET_IDS=$(aws cloudformation describe-stacks \
  --stack-name "$STACK_NAME" \
  --query "Stacks[0].Outputs[?OutputKey=='SubnetIds'].OutputValue" \
  --output text)
if [ -z "$SUBNET_IDS" ]; then
    handle_error "Failed to get Subnet IDs from CloudFormation stack"
fi
echo "Subnet IDs: $SUBNET_IDS"

SECURITY_GROUP_ID=$(aws cloudformation describe-stacks \
  --stack-name "$STACK_NAME" \
  --query "Stacks[0].Outputs[?OutputKey=='SecurityGroups'].OutputValue" \
  --output text)
if [ -z "$SECURITY_GROUP_ID" ]; then
    handle_error "Failed to get Security Group ID from CloudFormation stack"
fi
echo "Security Group ID: $SECURITY_GROUP_ID"
```

```
# Step 4: Create EKS cluster
echo "Step 4: Creating EKS cluster: $CLUSTER_NAME"

CLUSTER_ROLE_ARN=$(aws iam get-role --role-name "$CLUSTER_ROLE_NAME" --query
 "Role.Arn" --output text)
if [ -z "$CLUSTER_ROLE_ARN" ]; then
    handle_error "Failed to get Cluster Role ARN"
fi

echo "Creating EKS cluster (this will take 10-15 minutes)..."
CREATE_CLUSTER_OUTPUT=$(aws eks create-cluster \
  --name "$CLUSTER_NAME" \
  --role-arn "$CLUSTER_ROLE_ARN" \
  --resources-vpc-config
 subnetIds="$SUBNET_IDS",securityGroupIds="$SECURITY_GROUP_ID" \
  --tags Key=project,Value=doc-smith,Key=tutorial,Value=eks-gs)
check_command "$CREATE_CLUSTER_OUTPUT"
CREATED_RESOURCES+=("EKS Cluster: $CLUSTER_NAME")

echo "Waiting for EKS cluster to become active (this may take 10-15 minutes)..."
aws eks wait cluster-active --name "$CLUSTER_NAME"
if [ $? -ne 0 ]; then
    handle_error "Cluster creation failed or timed out"
fi
echo "EKS cluster is now active."

# Step 5: Configure kubectl
echo "Step 5: Configuring kubectl to communicate with the cluster..."

# Check if kubectl is installed
if ! check_kubectl; then
    echo "Will skip kubectl configuration steps but continue with the script."
    echo "You can manually configure kubectl later with: aws eks update-
kubeconfig --name \"$CLUSTER_NAME\""
else
    UPDATE_KUBECONFIG_OUTPUT=$(aws eks update-kubeconfig --name "$CLUSTER_NAME")
    check_command "$UPDATE_KUBECONFIG_OUTPUT"
    echo "kubectl configured successfully."

    # Test kubectl configuration
    echo "Testing kubectl configuration..."
    KUBECTL_TEST_OUTPUT=$(kubectl get svc 2>&1)
    if [ $? -ne 0 ]; then
```

```
        echo "Warning: kubectl configuration test failed. This might be due to
 permissions or network issues."
        echo "Error details: $KUBECTL_TEST_OUTPUT"
        echo "Continuing with script execution..."
    else
        echo "$KUBECTL_TEST_OUTPUT"
        echo "kubectl configuration test successful."
    fi
fi

# Step 6: Create managed node group
echo "Step 6: Creating managed node group: $NODEGROUP_NAME"

NODE_ROLE_ARN=$(aws iam get-role --role-name "$NODE_ROLE_NAME" --query "Role.Arn"
 --output text)
if [ -z "$NODE_ROLE_ARN" ]; then
    handle_error "Failed to get Node Role ARN"
fi

# Convert comma-separated subnet IDs to space-separated for the create-nodegroup
 command
SUBNET_IDS_ARRAY=(${SUBNET_IDS//,/ })

echo "Creating managed node group (this will take 5-10 minutes)..."
CREATE_NODEGROUP_OUTPUT=$(aws eks create-nodegroup \
  --cluster-name "$CLUSTER_NAME" \
  --nodegroup-name "$NODEGROUP_NAME" \
  --node-role "$NODE_ROLE_ARN" \
  --subnets "${SUBNET_IDS_ARRAY[@]}" \
  --tags Key=project,Value=doc-smith,Key=tutorial,Value=eks-gs)
check_command "$CREATE_NODEGROUP_OUTPUT"
CREATED_RESOURCES+=("EKS Node Group: $NODEGROUP_NAME")

echo "Waiting for node group to become active (this may take 5-10 minutes)..."
aws eks wait nodegroup-active --cluster-name "$CLUSTER_NAME" --nodegroup-name
 "$NODEGROUP_NAME"
if [ $? -ne 0 ]; then
    handle_error "Node group creation failed or timed out"
fi
echo "Node group is now active."

# Step 7: Verify nodes
echo "Step 7: Verifying nodes..."
```

```
echo "Waiting for nodes to register with the cluster (this may take a few
 minutes)..."
sleep 60  # Give nodes more time to register

# Check if kubectl is installed before attempting to use it
if ! check_kubectl; then
    echo "Cannot verify nodes without kubectl. Skipping this step."
    echo "You can manually verify nodes after installing kubectl with: kubectl
 get nodes"
else
    NODES_OUTPUT=$(kubectl get nodes 2>&1)
    if [ $? -ne 0 ]; then
        echo "Warning: Unable to get nodes. This might be due to permissions or
 the nodes are still registering."
        echo "Error details: $NODES_OUTPUT"
        echo "Continuing with script execution..."
    else
        echo "$NODES_OUTPUT"
        echo "Nodes verified successfully."
    fi
fi

# Step 8: View resources
echo "Step 8: Viewing cluster resources..."

echo "Cluster information:"
CLUSTER_INFO=$(aws eks describe-cluster --name "$CLUSTER_NAME")
echo "$CLUSTER_INFO"

echo "Node group information:"
NODEGROUP_INFO=$(aws eks describe-nodegroup --cluster-name "$CLUSTER_NAME" --
nodegroup-name "$NODEGROUP_NAME")
echo "$NODEGROUP_INFO"

echo "Kubernetes resources:"
if ! check_kubectl; then
    echo "Cannot list Kubernetes resources without kubectl. Skipping this step."
    echo "You can manually list resources after installing kubectl with: kubectl
 get all --all-namespaces"
else
    KUBE_RESOURCES=$(kubectl get all --all-namespaces 2>&1)
    if [ $? -ne 0 ]; then
        echo "Warning: Unable to get Kubernetes resources. This might be due to
 permissions."
```

```
        echo "Error details: $KUBE_RESOURCES"
        echo "Continuing with script execution..."
    else
        echo "$KUBE_RESOURCES"
    fi
fi

# Display summary of created resources
echo ""
echo "==========================================="
echo "RESOURCES CREATED"
echo "==========================================="
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "- $resource"
done
echo "==========================================="

# Prompt for cleanup
echo ""
echo "==========================================="
echo "CLEANUP CONFIRMATION"
echo "==========================================="
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    cleanup_resources
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later."
    echo "To clean up resources, run the following commands:"
    echo "1. Delete node group: aws eks delete-nodegroup --cluster-name
 $CLUSTER_NAME --nodegroup-name $NODEGROUP_NAME"
    echo "2. Wait for node group deletion: aws eks wait nodegroup-deleted --
cluster-name $CLUSTER_NAME --nodegroup-name $NODEGROUP_NAME"
    echo "3. Delete cluster: aws eks delete-cluster --name $CLUSTER_NAME"
    echo "4. Wait for cluster deletion: aws eks wait cluster-deleted --name
 $CLUSTER_NAME"
    echo "5. Delete CloudFormation stack: aws cloudformation delete-stack --
stack-name $STACK_NAME"
    echo "6. Detach and delete IAM roles for the node group and cluster"
fi

echo "Script completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateCluster
  - CreateNodegroup
  - CreateRole
  - CreateStack
  - DeleteCluster
  - DeleteNodegroup
  - DeleteRole
  - DeleteStack
  - DescribeCluster
  - DescribeNodegroup
  - DescribeStacks
  - DetachRolePolicy
  - GetCallerIdentity
  - GetRole
  - ListNodegroups
  - UpdateKubeconfig
  - Wait (CloudFormation)
  - Wait (Amazon EKS)

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with managed streaming

The following code example shows how to:

- Create an MSK cluster
- Create IAM permissions for MSK access

- Create a client machine
- Get bootstrap brokers
- Set up the client machine
- Create a topic and produce/consume data
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon MSK Getting Started Tutorial Script - Version 8
# This script automates the steps in the Amazon MSK Getting Started tutorial
# It creates an MSK cluster, sets up IAM permissions, creates a client machine,
# and configures the client to interact with the cluster

# Set up logging
LOG_FILE="msk_tutorial_$(date +%Y%m%d_%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon MSK Getting Started Tutorial Script - Version 8"
echo "Logging to $LOG_FILE"
echo "================================================"

# Function to handle errors
handle_error() {
    echo "ERROR: $1"
    echo "Resources created so far:"
    if [ -n "$CLUSTER_ARN" ]; then echo "- MSK Cluster: $CLUSTER_ARN"; fi
    if [ -n "$POLICY_ARN" ]; then echo "- IAM Policy: $POLICY_ARN"; fi
    if [ -n "$ROLE_NAME" ]; then echo "- IAM Role: $ROLE_NAME"; fi
    if [ -n "$INSTANCE_PROFILE_NAME" ]; then echo "- IAM Instance Profile:
 $INSTANCE_PROFILE_NAME"; fi
```

```
    if [ -n "$CLIENT_SG_ID" ]; then echo "- Client Security Group:
 $CLIENT_SG_ID"; fi
    if [ -n "$INSTANCE_ID" ]; then echo "- EC2 Instance: $INSTANCE_ID"; fi
    if [ -n "$KEY_NAME" ]; then echo "- Key Pair: $KEY_NAME"; fi

    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Function to check if a resource exists
resource_exists() {
    local resource_type="$1"
    local resource_id="$2"

    case "$resource_type" in
        "cluster")
            aws kafka describe-cluster --cluster-arn "$resource_id" &>/dev/null
            ;;
        "policy")
            aws iam get-policy --policy-arn "$resource_id" &>/dev/null
            ;;
        "role")
            aws iam get-role --role-name "$resource_id" &>/dev/null
            ;;
        "instance-profile")
            aws iam get-instance-profile --instance-profile-name "$resource_id"
 &>/dev/null
            ;;
        "security-group")
            aws ec2 describe-security-groups --group-ids "$resource_id" &>/dev/
null
            ;;
        "instance")
            aws ec2 describe-instances --instance-ids "$resource_id" --query
 'Reservations[0].Instances[0].State.Name' --output text | grep -v "terminated"
 &>/dev/null
            ;;
        "key-pair")
            aws ec2 describe-key-pairs --key-names "$resource_id" &>/dev/null
            ;;
    esac
}
```

```
# Function to remove security group references
remove_security_group_references() {
    local sg_id="$1"

    if [ -z "$sg_id" ]; then
        echo "No security group ID provided for reference removal"
        return
    fi

    echo "Removing security group references for $sg_id"

    # Get all security groups in the VPC that might reference our client security
 group
    local vpc_security_groups=$(aws ec2 describe-security-groups \
        --filters "Name=vpc-id,Values=$DEFAULT_VPC_ID" \
        --query 'SecurityGroups[].GroupId' \
        --output text 2>/dev/null)

    if [ -n "$vpc_security_groups" ]; then
        for other_sg in $vpc_security_groups; do
            if [ "$other_sg" != "$sg_id" ]; then
                echo "Checking security group $other_sg for references to $sg_id"

                # Get the security group details in JSON format
                local sg_details=$(aws ec2 describe-security-groups \
                    --group-ids "$other_sg" \
                    --output json 2>/dev/null)

                if [ -n "$sg_details" ]; then
                    # Check if our security group is referenced in inbound rules
                    local has_inbound_ref=$(echo "$sg_details" | grep -o
 "\"GroupId\": \"$sg_id\"" | head -1)

                    if [ -n "$has_inbound_ref" ]; then
                        echo "Found inbound rules in $other_sg referencing
 $sg_id, removing them..."

                        # Try to remove common rule types
                        echo "Attempting to remove all-traffic rule"
                        aws ec2 revoke-security-group-ingress \
                            --group-id "$other_sg" \
                            --protocol all \
                            --source-group "$sg_id" 2>/dev/null || echo "No all-
traffic rule to remove"
```

```
                        # Try to remove TCP rules on common ports
                        for port in 22 80 443 9092 9094 9096; do
                            aws ec2 revoke-security-group-ingress \
                                --group-id "$other_sg" \
                                --protocol tcp \
                                --port "$port" \
                                --source-group "$sg_id" 2>/dev/null || true
                        done

                        # Try to remove UDP rules
                        aws ec2 revoke-security-group-ingress \
                            --group-id "$other_sg" \
                            --protocol udp \
                            --source-group "$sg_id" 2>/dev/null || true
                    fi

                    # Check for outbound rules (less common but possible)
                    local has_outbound_ref=$(echo "$sg_details" | grep -A 20
 "IpPermissionsEgress" | grep -o "\"GroupId\": \"$sg_id\"" | head -1)

                    if [ -n "$has_outbound_ref" ]; then
                        echo "Found outbound rules in $other_sg referencing
 $sg_id, removing them..."

                        aws ec2 revoke-security-group-egress \
                            --group-id "$other_sg" \
                            --protocol all \
                            --source-group "$sg_id" 2>/dev/null || echo "No
 outbound all-traffic rule to remove"
                    fi
                fi
            fi
        done
    fi

    echo "Completed security group reference removal for $sg_id"
}

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources..."

    # Delete EC2 instance if it exists
```

```
   if [ -n "$INSTANCE_ID" ] && resource_exists "instance" "$INSTANCE_ID"; then
       echo "Terminating EC2 instance: $INSTANCE_ID"
       aws ec2 terminate-instances --instance-ids "$INSTANCE_ID" || echo "Failed
to terminate instance"
       echo "Waiting for instance to terminate..."
       aws ec2 wait instance-terminated --instance-ids "$INSTANCE_ID" || echo
"Failed to wait for instance termination"
   fi

   # Delete MSK cluster first (to remove dependencies on security group)
   if [ -n "$CLUSTER_ARN" ] && resource_exists "cluster" "$CLUSTER_ARN"; then
       echo "Deleting MSK cluster: $CLUSTER_ARN"
       aws kafka delete-cluster --cluster-arn "$CLUSTER_ARN" || echo "Failed to
delete cluster"

       # Wait a bit for the cluster deletion to start
       echo "Waiting 30 seconds for cluster deletion to begin..."
       sleep 30
   fi

   # Remove security group references before attempting deletion
   if [ -n "$CLIENT_SG_ID" ] && resource_exists "security-group"
"$CLIENT_SG_ID"; then
       remove_security_group_references "$CLIENT_SG_ID"

       echo "Deleting security group: $CLIENT_SG_ID"
       # Try multiple times with longer delays to ensure dependencies are
removed
       for i in {1..10}; do
           if aws ec2 delete-security-group --group-id "$CLIENT_SG_ID"; then
               echo "Security group deleted successfully"
               break
           fi
           echo "Failed to delete security group (attempt $i/10), retrying in 30
seconds..."
           sleep 30
       done
   fi

   # Delete key pair if it exists
   if [ -n "$KEY_NAME" ] && resource_exists "key-pair" "$KEY_NAME"; then
       echo "Deleting key pair: $KEY_NAME"
       aws ec2 delete-key-pair --key-name "$KEY_NAME" || echo "Failed to delete
key pair"
```

```
    fi

    # Remove role from instance profile
    if [ -n "$ROLE_NAME" ] && [ -n "$INSTANCE_PROFILE_NAME" ] && resource_exists
"instance-profile" "$INSTANCE_PROFILE_NAME"; then
        echo "Removing role from instance profile"
        aws iam remove-role-from-instance-profile \
            --instance-profile-name "$INSTANCE_PROFILE_NAME" \
            --role-name "$ROLE_NAME" || echo "Failed to remove role from instance
profile"
    fi

    # Delete instance profile
    if [ -n "$INSTANCE_PROFILE_NAME" ] && resource_exists "instance-profile"
"$INSTANCE_PROFILE_NAME"; then
        echo "Deleting instance profile: $INSTANCE_PROFILE_NAME"
        aws iam delete-instance-profile \
            --instance-profile-name "$INSTANCE_PROFILE_NAME" || echo "Failed to
delete instance profile"
    fi

    # Detach policy from role
    if [ -n "$ROLE_NAME" ] && [ -n "$POLICY_ARN" ] && resource_exists "role"
"$ROLE_NAME"; then
        echo "Detaching policy from role"
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "$POLICY_ARN" || echo "Failed to detach policy"
    fi

    # Delete role
    if [ -n "$ROLE_NAME" ] && resource_exists "role" "$ROLE_NAME"; then
        echo "Deleting role: $ROLE_NAME"
        aws iam delete-role --role-name "$ROLE_NAME" || echo "Failed to delete
role"
    fi

    # Delete policy
    if [ -n "$POLICY_ARN" ] && resource_exists "policy" "$POLICY_ARN"; then
        echo "Deleting policy: $POLICY_ARN"
        aws iam delete-policy --policy-arn "$POLICY_ARN" || echo "Failed to
delete policy"
    fi
```

```
    echo "Cleanup completed"
}

# Function to find a suitable subnet and instance type combination
find_suitable_subnet_and_instance_type() {
    local vpc_id="$1"
    local -a subnet_array=("${!2}")

    # List of instance types to try, in order of preference
    local instance_types=("t3.micro" "t2.micro" "t3.small" "t2.small")

    echo "Finding suitable subnet and instance type combination..."

    for instance_type in "${instance_types[@]}"; do
        echo "Trying instance type: $instance_type"

        for subnet_id in "${subnet_array[@]}"; do
            # Get the availability zone for this subnet
            local az=$(aws ec2 describe-subnets \
                --subnet-ids "$subnet_id" \
                --query 'Subnets[0].AvailabilityZone' \
                --output text)

            echo "  Checking subnet $subnet_id in AZ $az"

            # Check if this instance type is available in this AZ
            local available=$(aws ec2 describe-instance-type-offerings \
                --location-type availability-zone \
                --filters "Name=location,Values=$az" "Name=instance-type,Values=
$instance_type" \
                --query 'InstanceTypeOfferings[0].InstanceType' \
                --output text 2>/dev/null)

            if [ "$available" = "$instance_type" ]; then
                echo "  # Found suitable combination: $instance_type in $az
 (subnet: $subnet_id)"
                SELECTED_SUBNET_ID="$subnet_id"
                SELECTED_INSTANCE_TYPE="$instance_type"
                return 0
            else
                echo "  # $instance_type not available in $az"
            fi
        done
    done
```

```
    echo "ERROR: Could not find any suitable subnet and instance type
 combination"
    return 1
}

# Generate unique identifiers
RANDOM_SUFFIX=$(LC_ALL=C tr -dc 'a-z0-9' < /dev/urandom | fold -w 8 | head -n 1)
CLUSTER_NAME="MSKTutorialCluster-${RANDOM_SUFFIX}"
POLICY_NAME="msk-tutorial-policy-${RANDOM_SUFFIX}"
ROLE_NAME="msk-tutorial-role-${RANDOM_SUFFIX}"
INSTANCE_PROFILE_NAME="msk-tutorial-profile-${RANDOM_SUFFIX}"
SG_NAME="MSKClientSecurityGroup-${RANDOM_SUFFIX}"

echo "Using the following resource names:"
echo "- Cluster Name: $CLUSTER_NAME"
echo "- Policy Name: $POLICY_NAME"
echo "- Role Name: $ROLE_NAME"
echo "- Instance Profile Name: $INSTANCE_PROFILE_NAME"
echo "- Security Group Name: $SG_NAME"
echo "================================================"

# Step 1: Create an MSK Provisioned cluster
echo "Step 1: Creating MSK Provisioned cluster"

# Get the default VPC ID first
echo "Getting default VPC..."
DEFAULT_VPC_ID=$(aws ec2 describe-vpcs \
    --filters "Name=is-default,Values=true" \
    --query "Vpcs[0].VpcId" \
    --output text)

if [ -z "$DEFAULT_VPC_ID" ] || [ "$DEFAULT_VPC_ID" = "None" ]; then
    handle_error "Could not find default VPC. Please ensure you have a default
 VPC in your region."
fi

echo "Default VPC ID: $DEFAULT_VPC_ID"

# Get available subnets in the default VPC
echo "Getting available subnets in the default VPC..."
SUBNETS=$(aws ec2 describe-subnets \
    --filters "Name=vpc-id,Values=$DEFAULT_VPC_ID" "Name=default-for-
az,Values=true" \
```

```
    --query "Subnets[0:3].SubnetId" \
    --output text)

# Convert space-separated subnet IDs to an array
read -r -a SUBNET_ARRAY <<< "$SUBNETS"

if [ ${#SUBNET_ARRAY[@]} -lt 3 ]; then
    handle_error "Not enough subnets available in the default VPC. Need at least
 3 subnets, found ${#SUBNET_ARRAY[@]}."
fi

# Get default security group for the default VPC
echo "Getting default security group for the default VPC..."
DEFAULT_SG=$(aws ec2 describe-security-groups \
    --filters "Name=group-name,Values=default" "Name=vpc-id,Values=
$DEFAULT_VPC_ID" \
    --query "SecurityGroups[0].GroupId" \
    --output text)

if [ -z "$DEFAULT_SG" ] || [ "$DEFAULT_SG" = "None" ]; then
    handle_error "Could not find default security group for VPC $DEFAULT_VPC_ID"
fi

echo "Creating MSK cluster: $CLUSTER_NAME"
echo "Using VPC: $DEFAULT_VPC_ID"
echo "Using subnets: ${SUBNET_ARRAY[0]} ${SUBNET_ARRAY[1]} ${SUBNET_ARRAY[2]}"
echo "Using security group: $DEFAULT_SG"

# Create the MSK cluster with proper error handling
CLUSTER_RESPONSE=$(aws kafka create-cluster \
    --cluster-name "$CLUSTER_NAME" \
    --broker-node-group-info "{\"InstanceType\": \"kafka.t3.small\",
 \"ClientSubnets\": [\"${SUBNET_ARRAY[0]}\", \"${SUBNET_ARRAY[1]}\",
 \"${SUBNET_ARRAY[2]}\"], \"SecurityGroups\": [\"$DEFAULT_SG\"]}" \
    --kafka-version "3.6.0" \
    --number-of-broker-nodes 3 \
    --encryption-info "{\"EncryptionInTransit\": {\"InCluster\": true,
 \"ClientBroker\": \"TLS\"}}" \
    --tags project=doc-smith,tutorial=amazon-managed-streaming-for-apache-kafka-
gs 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create MSK cluster: $CLUSTER_RESPONSE"
```

```
fi

# Extract the cluster ARN using grep
CLUSTER_ARN=$(echo "$CLUSTER_RESPONSE" | grep -o '"ClusterArn": "[^"]*' | cut -
d'"' -f4)

if [ -z "$CLUSTER_ARN" ]; then
    handle_error "Failed to extract cluster ARN from response: $CLUSTER_RESPONSE"
fi

echo "MSK cluster creation initiated. ARN: $CLUSTER_ARN"
echo "Waiting for cluster to become active (this may take 15-20 minutes)..."

# Wait for the cluster to become active
while true; do
    CLUSTER_STATUS=$(aws kafka describe-cluster --cluster-arn "$CLUSTER_ARN" --
query "ClusterInfo.State" --output text 2>/dev/null)

    if [ $? -ne 0 ]; then
        echo "Failed to get cluster status. Retrying in 30 seconds..."
        sleep 30
        continue
    fi

    echo "Current cluster status: $CLUSTER_STATUS"

    if [ "$CLUSTER_STATUS" = "ACTIVE" ]; then
        echo "Cluster is now active!"
        break
    elif [ "$CLUSTER_STATUS" = "FAILED" ]; then
        handle_error "Cluster creation failed"
    fi

    echo "Still waiting for cluster to become active... (checking again in 60
 seconds)"
    sleep 60
done

echo "============================================="
# Step 2: Create an IAM role granting access to create topics on the Amazon MSK
 cluster
echo "Step 2: Creating IAM policy and role"

# Get account ID and region
```

```
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
REGION=$(aws configure get region)
if [ -z "$REGION" ]; then
    REGION=$(aws ec2 describe-availability-zones --query
 'AvailabilityZones[0].RegionName' --output text)
fi

if [ -z "$ACCOUNT_ID" ] || [ -z "$REGION" ]; then
    handle_error "Could not determine AWS account ID or region"
fi

echo "Account ID: $ACCOUNT_ID"
echo "Region: $REGION"

# Create IAM policy
echo "Creating IAM policy: $POLICY_NAME"
POLICY_DOCUMENT="{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
        {
            \"Effect\": \"Allow\",
            \"Action\": [
                \"kafka-cluster:Connect\",
                \"kafka-cluster:AlterCluster\",
                \"kafka-cluster:DescribeCluster\"
            ],
            \"Resource\": [
                \"$CLUSTER_ARN\"
            ]
        },
        {
            \"Effect\": \"Allow\",
            \"Action\": [
                \"kafka-cluster:*Topic*\",
                \"kafka-cluster:WriteData\",
                \"kafka-cluster:ReadData\"
            ],
            \"Resource\": [
                \"arn:aws:kafka:$REGION:$ACCOUNT_ID:topic/$CLUSTER_NAME/*\"
            ]
        },
        {
            \"Effect\": \"Allow\",
            \"Action\": [
```

```
                \"kafka-cluster:AlterGroup\",
                \"kafka-cluster:DescribeGroup\"
            ],
            \"Resource\": [
                \"arn:aws:kafka:$REGION:$ACCOUNT_ID:group/$CLUSTER_NAME/*\"
            ]
        }
    ]
}"

POLICY_RESPONSE=$(aws iam create-policy \
    --policy-name "$POLICY_NAME" \
    --policy-document "$POLICY_DOCUMENT" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create IAM policy: $POLICY_RESPONSE"
fi

# Extract the policy ARN using grep
POLICY_ARN=$(echo "$POLICY_RESPONSE" | grep -o '"Arn": "[^"]*' | cut -d'"' -f4)

if [ -z "$POLICY_ARN" ]; then
    handle_error "Failed to extract policy ARN from response: $POLICY_RESPONSE"
fi

aws iam tag-policy --policy-arn "$POLICY_ARN" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=amazon-managed-
streaming-for-apache-kafka-gs

echo "IAM policy created. ARN: $POLICY_ARN"

# Create IAM role for EC2
echo "Creating IAM role: $ROLE_NAME"
TRUST_POLICY="{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow\",
\"Principal\":{\"Service\":\"ec2.amazonaws.com\"},\"Action\":\"sts:AssumeRole
\"}]}"

ROLE_RESPONSE=$(aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document "$TRUST_POLICY" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
```

```
    handle_error "Failed to create IAM role: $ROLE_RESPONSE"
fi

echo "IAM role created: $ROLE_NAME"
aws iam tag-role --role-name "$ROLE_NAME" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=amazon-managed-
streaming-for-apache-kafka-gs

# Attach policy to role
echo "Attaching policy to role"
ATTACH_RESPONSE=$(aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "$POLICY_ARN" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to attach policy to role: $ATTACH_RESPONSE"
fi

echo "Policy attached to role"

# Create instance profile and add role to it
echo "Creating instance profile: $INSTANCE_PROFILE_NAME"
PROFILE_RESPONSE=$(aws iam create-instance-profile \
    --instance-profile-name "$INSTANCE_PROFILE_NAME" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create instance profile: $PROFILE_RESPONSE"
fi

echo "Instance profile created"

echo "Adding role to instance profile"
ADD_ROLE_RESPONSE=$(aws iam add-role-to-instance-profile \
    --instance-profile-name "$INSTANCE_PROFILE_NAME" \
    --role-name "$ROLE_NAME" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to add role to instance profile: $ADD_ROLE_RESPONSE"
fi

echo "Role added to instance profile"
```

```
# Wait a moment for IAM propagation
echo "Waiting 10 seconds for IAM propagation..."
sleep 10

echo "============================================="

# Step 3: Create a client machine
echo "Step 3: Creating client machine"

# Find a suitable subnet and instance type combination
if ! find_suitable_subnet_and_instance_type "$DEFAULT_VPC_ID" SUBNET_ARRAY[@];
 then
    handle_error "Could not find a suitable subnet and instance type combination"
fi

echo "Selected subnet: $SELECTED_SUBNET_ID"
echo "Selected instance type: $SELECTED_INSTANCE_TYPE"

# Verify the subnet is in the same VPC we're using
SUBNET_VPC_ID=$(aws ec2 describe-subnets \
    --subnet-ids "$SELECTED_SUBNET_ID" \
    --query 'Subnets[0].VpcId' \
    --output text)

if [ "$SUBNET_VPC_ID" != "$DEFAULT_VPC_ID" ]; then
    handle_error "Subnet VPC ($SUBNET_VPC_ID) does not match default VPC
 ($DEFAULT_VPC_ID)"
fi

echo "VPC ID: $SUBNET_VPC_ID"

# Get security group ID from the MSK cluster
echo "Getting security group ID from the MSK cluster"
MSK_SG_ID=$(aws kafka describe-cluster \
    --cluster-arn "$CLUSTER_ARN" \
    --query 'ClusterInfo.BrokerNodeGroupInfo.SecurityGroups[0]' \
    --output text)

if [ -z "$MSK_SG_ID" ] || [ "$MSK_SG_ID" = "None" ]; then
    handle_error "Failed to get security group ID from cluster"
fi

echo "MSK security group ID: $MSK_SG_ID"
```

```
# Create security group for client
echo "Creating security group for client: $SG_NAME"
CLIENT_SG_RESPONSE=$(aws ec2 create-security-group \
    --group-name "$SG_NAME" \
    --description "Security group for MSK client" \
    --vpc-id "$DEFAULT_VPC_ID" \
    --tag-specifications 'ResourceType=security-
group,Tags=[{Key=project,Value=doc-smith},{Key=tutorial,Value=amazon-managed-
streaming-for-apache-kafka-gs}]' 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create security group: $CLIENT_SG_RESPONSE"
fi

# Extract the security group ID using grep
CLIENT_SG_ID=$(echo "$CLIENT_SG_RESPONSE" | grep -o '"GroupId": "[^"]*' | cut -
d'"' -f4)

if [ -z "$CLIENT_SG_ID" ]; then
    handle_error "Failed to extract security group ID from response:
 $CLIENT_SG_RESPONSE"
fi

echo "Client security group created. ID: $CLIENT_SG_ID"

# Allow SSH access to client from your IP only
echo "Getting your public IP address"
MY_IP=$(curl -s https://checkip.amazonaws.com 2>/dev/null)

if [ -z "$MY_IP" ]; then
    echo "Warning: Could not determine your IP address. Using 0.0.0.0/0 (not
 recommended for production)"
    MY_IP="0.0.0.0/0"
else
    MY_IP="$MY_IP/32"
    echo "Your IP address: $MY_IP"
fi

echo "Adding SSH ingress rule to client security group"
SSH_RULE_RESPONSE=$(aws ec2 authorize-security-group-ingress \
    --group-id "$CLIENT_SG_ID" \
    --protocol tcp \
```

```
    --port 22 \
    --cidr "$MY_IP" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    echo "Warning: Failed to add SSH ingress rule: $SSH_RULE_RESPONSE"
    echo "You may need to manually add SSH access to security group
 $CLIENT_SG_ID"
fi

echo "SSH ingress rule added"

# Update MSK security group to allow traffic from client security group
echo "Adding ingress rule to MSK security group to allow traffic from client"
MSK_RULE_RESPONSE=$(aws ec2 authorize-security-group-ingress \
    --group-id "$MSK_SG_ID" \
    --protocol all \
    --source-group "$CLIENT_SG_ID" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    echo "Warning: Failed to add ingress rule to MSK security group:
 $MSK_RULE_RESPONSE"
    echo "You may need to manually add ingress rule to security group $MSK_SG_ID"
fi

echo "Ingress rule added to MSK security group"

# Create key pair
KEY_NAME="MSKKeyPair-${RANDOM_SUFFIX}"
echo "Creating key pair: $KEY_NAME"
KEY_RESPONSE=$(aws ec2 create-key-pair --key-name "$KEY_NAME" --tag-
specifications 'ResourceType=key-pair,Tags=[{Key=project,Value=doc-smith},
{Key=tutorial,Value=amazon-managed-streaming-for-apache-kafka-gs}]' --query
 'KeyMaterial' --output text 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to create key pair: $KEY_RESPONSE"
fi

# Save the private key to a file
KEY_FILE="${KEY_NAME}.pem"
echo "$KEY_RESPONSE" > "$KEY_FILE"
```

```
chmod 400 "$KEY_FILE"

echo "Key pair created and saved to $KEY_FILE"

# Get the latest Amazon Linux 2 AMI
echo "Getting latest Amazon Linux 2 AMI ID"
AMI_ID=$(aws ec2 describe-images \
    --owners amazon \
    --filters "Name=name,Values=amzn2-ami-hvm-*-x86_64-gp2"
 "Name=state,Values=available" \
    --query "sort_by(Images, &CreationDate)[-1].ImageId" \
    --output text 2>/dev/null)

if [ -z "$AMI_ID" ] || [ "$AMI_ID" = "None" ]; then
    handle_error "Failed to get Amazon Linux 2 AMI ID"
fi

echo "Using AMI ID: $AMI_ID"

# Launch EC2 instance with the selected subnet and instance type
echo "Launching EC2 instance"
echo "Instance type: $SELECTED_INSTANCE_TYPE"
echo "Subnet: $SELECTED_SUBNET_ID"

INSTANCE_RESPONSE=$(aws ec2 run-instances \
    --image-id "$AMI_ID" \
    --instance-type "$SELECTED_INSTANCE_TYPE" \
    --key-name "$KEY_NAME" \
    --security-group-ids "$CLIENT_SG_ID" \
    --subnet-id "$SELECTED_SUBNET_ID" \
    --iam-instance-profile "Name=$INSTANCE_PROFILE_NAME" \
    --tag-specifications
 "ResourceType=instance,Tags=[{Key=Name,Value=MSKTutorialClient-
${RANDOM_SUFFIX}},{Key=project,Value=doc-smith},{Key=tutorial,Value=amazon-
managed-streaming-for-apache-kafka-gs}]" 2>&1)

# Check if the command was successful
if [ $? -ne 0 ]; then
    handle_error "Failed to launch EC2 instance: $INSTANCE_RESPONSE"
fi

# Extract the instance ID using grep
INSTANCE_ID=$(echo "$INSTANCE_RESPONSE" | grep -o '"InstanceId": "[^"]*' | head
 -1 | cut -d'"' -f4)
```

```
if [ -z "$INSTANCE_ID" ]; then
    handle_error "Failed to extract instance ID from response:
 $INSTANCE_RESPONSE"
fi

echo "EC2 instance launched successfully. ID: $INSTANCE_ID"
echo "Waiting for instance to be running..."

# Wait for the instance to be running
aws ec2 wait instance-running --instance-ids "$INSTANCE_ID"

if [ $? -ne 0 ]; then
    handle_error "Instance failed to reach running state"
fi

# Wait a bit more for the instance to initialize
echo "Instance is running. Waiting 30 seconds for initialization..."
sleep 30

# Get public DNS name of instance
CLIENT_DNS=$(aws ec2 describe-instances \
    --instance-ids "$INSTANCE_ID" \
    --query 'Reservations[0].Instances[0].PublicDnsName' \
    --output text)

if [ -z "$CLIENT_DNS" ] || [ "$CLIENT_DNS" = "None" ]; then
    echo "Warning: Could not get public DNS name for instance. Trying public
 IP..."
    CLIENT_DNS=$(aws ec2 describe-instances \
        --instance-ids "$INSTANCE_ID" \
        --query 'Reservations[0].Instances[0].PublicIpAddress' \
        --output text)

    if [ -z "$CLIENT_DNS" ] || [ "$CLIENT_DNS" = "None" ]; then
        handle_error "Failed to get public DNS name or IP address for instance"
    fi
fi

echo "Client instance DNS/IP: $CLIENT_DNS"
echo "============================================="
# Get bootstrap brokers with improved logic
echo "Getting bootstrap brokers"
MAX_RETRIES=10
```

```
RETRY_COUNT=0
BOOTSTRAP_BROKERS=""
AUTH_METHOD=""

while [ -z "$BOOTSTRAP_BROKERS" ] || [ "$BOOTSTRAP_BROKERS" = "None" ]; do
    # Get the full bootstrap brokers response
    BOOTSTRAP_RESPONSE=$(aws kafka get-bootstrap-brokers \
        --cluster-arn "$CLUSTER_ARN" 2>/dev/null)

    if [ $? -eq 0 ] && [ -n "$BOOTSTRAP_RESPONSE" ]; then
        # Try to get IAM authentication brokers first using grep
        BOOTSTRAP_BROKERS=$(echo "$BOOTSTRAP_RESPONSE" | grep -o
 '"BootstrapBrokerStringSaslIam": "[^"]*' | cut -d'"' -f4)
        if [ -n "$BOOTSTRAP_BROKERS" ]; then
            AUTH_METHOD="IAM"
        else
            # Fall back to TLS authentication
            BOOTSTRAP_BROKERS=$(echo "$BOOTSTRAP_RESPONSE" | grep -o
 '"BootstrapBrokerStringTls": "[^"]*' | cut -d'"' -f4)
            if [ -n "$BOOTSTRAP_BROKERS" ]; then
                AUTH_METHOD="TLS"
            fi
        fi
    fi

    RETRY_COUNT=$((RETRY_COUNT + 1))

    if [ "$RETRY_COUNT" -ge "$MAX_RETRIES" ]; then
        echo "Warning: Could not get bootstrap brokers after $MAX_RETRIES
 attempts."
        echo "You may need to manually retrieve them later using:"
        echo "aws kafka get-bootstrap-brokers --cluster-arn $CLUSTER_ARN"
        BOOTSTRAP_BROKERS="BOOTSTRAP_BROKERS_NOT_AVAILABLE"
        AUTH_METHOD="UNKNOWN"
        break
    fi

    if [ -z "$BOOTSTRAP_BROKERS" ] || [ "$BOOTSTRAP_BROKERS" = "None" ]; then
        echo "Bootstrap brokers not available yet. Retrying in 30 seconds...
 (Attempt $RETRY_COUNT/$MAX_RETRIES)"
        sleep 30
    fi
done
```

```
echo "Bootstrap brokers: $BOOTSTRAP_BROKERS"
echo "Authentication method: $AUTH_METHOD"
echo "============================================="

# Create setup script for the client machine
echo "Creating setup script for the client machine"
cat > setup_client.sh << 'EOF'
#!/bin/bash

# Set up logging
LOG_FILE="client_setup_$(date +%Y%m%d_%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting client setup"
echo "============================================="

# Install Java
echo "Installing Java"
sudo yum -y install java-11

# Set environment variables
echo "Setting up environment variables"
export KAFKA_VERSION="3.6.0"
echo "KAFKA_VERSION=$KAFKA_VERSION"

# Download and extract Apache Kafka
echo "Downloading Apache Kafka"
wget https://archive.apache.org/dist/kafka/$KAFKA_VERSION/kafka_2.13-
$KAFKA_VERSION.tgz
if [ $? -ne 0 ]; then
    echo "Failed to download Kafka. Trying alternative mirror..."
    wget https://www.apache.org/dyn/closer.cgi?path=/kafka/$KAFKA_VERSION/
kafka_2.13-$KAFKA_VERSION.tgz
fi

echo "Extracting Kafka"
tar -xzf kafka_2.13-$KAFKA_VERSION.tgz
export KAFKA_ROOT=$(pwd)/kafka_2.13-$KAFKA_VERSION
echo "KAFKA_ROOT=$KAFKA_ROOT"

# Download the MSK IAM authentication package (needed for both IAM and TLS)
echo "Downloading MSK IAM authentication package"
cd $KAFKA_ROOT/libs
```

```
wget https://github.com/aws/aws-msk-iam-auth/releases/latest/download/aws-msk-
iam-auth-1.1.6-all.jar
if [ $? -ne 0 ]; then
    echo "Failed to download specific version. Trying to get latest version..."
    LATEST_VERSION=$(curl -s https://api.github.com/repos/aws/aws-msk-iam-auth/
releases/latest | grep -o '"tag_name": "[^"]*' | cut -d'"' -f4)
    wget https://github.com/aws/aws-msk-iam-auth/releases/download/
$LATEST_VERSION/aws-msk-iam-auth-$LATEST_VERSION-all.jar
    if [ $? -ne 0 ]; then
        echo "Failed to download IAM auth package. Please check the URL and try
 again."
        exit 1
    fi
    export CLASSPATH=$KAFKA_ROOT/libs/aws-msk-iam-auth-$LATEST_VERSION-all.jar
else
    export CLASSPATH=$KAFKA_ROOT/libs/aws-msk-iam-auth-1.1.6-all.jar
fi
echo "CLASSPATH=$CLASSPATH"

# Create client properties file based on authentication method
echo "Creating client properties file"
cd $KAFKA_ROOT/config

# The AUTH_METHOD_PLACEHOLDER will be replaced by the script
AUTH_METHOD="AUTH_METHOD_PLACEHOLDER"

if [ "$AUTH_METHOD" = "IAM" ]; then
    echo "Configuring for IAM authentication"
    cat > client.properties << 'EOT'
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=software.amazon.msk.auth.iam.IAMLoginModule required;
sasl.client.callback.handler.class=software.amazon.msk.auth.iam.IAMClientCallbackHandler
EOT
elif [ "$AUTH_METHOD" = "TLS" ]; then
    echo "Configuring for TLS authentication"
    cat > client.properties << 'EOT'
security.protocol=SSL
EOT
else
    echo "Unknown authentication method. Creating basic TLS configuration."
    cat > client.properties << 'EOT'
security.protocol=SSL
EOT
```

```
fi

echo "Client setup completed"
echo "============================================="

# Create a script to set up environment variables
cat > ~/setup_env.sh << 'EOT'
#!/bin/bash
export KAFKA_VERSION="3.6.0"
export KAFKA_ROOT=~/kafka_2.13-$KAFKA_VERSION
export CLASSPATH=$KAFKA_ROOT/libs/aws-msk-iam-auth-1.1.6-all.jar
export BOOTSTRAP_SERVER="BOOTSTRAP_SERVER_PLACEHOLDER"
export AUTH_METHOD="AUTH_METHOD_PLACEHOLDER"

echo "Environment variables set:"
echo "KAFKA_VERSION=$KAFKA_VERSION"
echo "KAFKA_ROOT=$KAFKA_ROOT"
echo "CLASSPATH=$CLASSPATH"
echo "BOOTSTRAP_SERVER=$BOOTSTRAP_SERVER"
echo "AUTH_METHOD=$AUTH_METHOD"
EOT

chmod +x ~/setup_env.sh

echo "Created environment setup script: ~/setup_env.sh"
echo "Run 'source ~/setup_env.sh' to set up your environment"
EOF

# Replace placeholders in the setup script
if [ -n "$BOOTSTRAP_BROKERS" ] && [ "$BOOTSTRAP_BROKERS" != "None" ] &&
 [ "$BOOTSTRAP_BROKERS" != "BOOTSTRAP_BROKERS_NOT_AVAILABLE" ]; then
    sed -i "s|BOOTSTRAP_SERVER_PLACEHOLDER|$BOOTSTRAP_BROKERS|g" setup_client.sh
else
    # If bootstrap brokers are not available, provide instructions to get them
    sed -i "s|BOOTSTRAP_SERVER_PLACEHOLDER|\$(aws kafka get-bootstrap-brokers
 --cluster-arn $CLUSTER_ARN --query 'BootstrapBrokerStringTls' --output text)|g"
 setup_client.sh
fi

# Replace auth method placeholder
sed -i "s|AUTH_METHOD_PLACEHOLDER|$AUTH_METHOD|g" setup_client.sh

echo "Setup script created"
echo "============================================="
```

```
# Display summary of created resources
echo ""
echo "================================================"
echo "RESOURCE SUMMARY"
echo "================================================"
echo "MSK Cluster ARN: $CLUSTER_ARN"
echo "MSK Cluster Name: $CLUSTER_NAME"
echo "Authentication Method: $AUTH_METHOD"
echo "IAM Policy ARN: $POLICY_ARN"
echo "IAM Role Name: $ROLE_NAME"
echo "IAM Instance Profile: $INSTANCE_PROFILE_NAME"
echo "Client Security Group: $CLIENT_SG_ID"
echo "EC2 Instance ID: $INSTANCE_ID"
echo "EC2 Instance Type: $SELECTED_INSTANCE_TYPE"
echo "EC2 Instance DNS: $CLIENT_DNS"
echo "Key Pair: $KEY_NAME (saved to $KEY_FILE)"
echo "Bootstrap Brokers: $BOOTSTRAP_BROKERS"
echo "================================================"
echo ""

# Instructions for connecting to the instance and setting up the client
echo "NEXT STEPS:"
echo "1. Connect to your EC2 instance:"
echo "   ssh -i $KEY_FILE ec2-user@$CLIENT_DNS"
echo ""
echo "2. Upload the setup script to your instance:"
echo "   scp -i $KEY_FILE setup_client.sh ec2-user@$CLIENT_DNS:~/"
echo ""
echo "3. Run the setup script on your instance:"
echo "   ssh -i $KEY_FILE ec2-user@$CLIENT_DNS 'chmod +x ~/setup_client.sh && ~/
setup_client.sh'"
echo ""
echo "4. Source the environment setup script:"
echo "   source ~/setup_env.sh"
echo ""

# Provide different instructions based on authentication method
if [ "$AUTH_METHOD" = "IAM" ]; then
    echo "5. Create a Kafka topic (using IAM authentication):"
    echo "   \$KAFKA_ROOT/bin/kafka-topics.sh --create \\"
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --command-config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --replication-factor 3 \\"
```

```
    echo "     --partitions 1 \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "6. Start a producer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-producer.sh \\"
    echo "     --broker-list \$BOOTSTRAP_SERVER \\"
    echo "     --producer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "7. Start a consumer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-consumer.sh \\"
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --consumer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic \\"
    echo "     --from-beginning"
elif [ "$AUTH_METHOD" = "TLS" ]; then
    echo "5. Create a Kafka topic (using TLS authentication):"
    echo "   \$KAFKA_ROOT/bin/kafka-topics.sh --create \\"
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --command-config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --replication-factor 3 \\"
    echo "     --partitions 1 \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "6. Start a producer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-producer.sh \\"
    echo "     --broker-list \$BOOTSTRAP_SERVER \\"
    echo "     --producer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic"
    echo ""
    echo "7. Start a consumer:"
    echo "   \$KAFKA_ROOT/bin/kafka-console-consumer.sh \\"
    echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
    echo "     --consumer.config \$KAFKA_ROOT/config/client.properties \\"
    echo "     --topic MSKTutorialTopic \\"
    echo "     --from-beginning"
else
    echo "5. Manually retrieve bootstrap brokers and configure authentication:"
    echo "   aws kafka get-bootstrap-brokers --cluster-arn $CLUSTER_ARN"
fi

echo ""
echo "8. Verify the topic was created:"
echo "   \$KAFKA_ROOT/bin/kafka-topics.sh --list \\"
```

```
echo "     --bootstrap-server \$BOOTSTRAP_SERVER \\"
echo "     --command-config \$KAFKA_ROOT/config/client.properties"
echo "================================================"
echo ""

# Ask if user wants to clean up resources
echo ""
echo "================================================"
echo "CLEANUP CONFIRMATION"
echo "================================================"
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ $CLEANUP_CHOICE =~ ^[Yy]$ ]]; then
    cleanup_resources
    echo "All resources have been cleaned up."
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later."
    echo "To clean up resources, run this script again and choose 'y' when
 prompted."
fi

echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddRoleToInstanceProfile
  - AttachRolePolicy
  - AuthorizeSecurityGroupIngress
  - CreateCluster
  - CreateInstanceProfile
  - CreateKeyPair
  - CreatePolicy
  - CreateRole
  - CreateSecurityGroup
  - DeleteCluster
  - DeleteInstanceProfile

- DeleteKeyPair
- DeletePolicy
- DeleteRole
- DeleteSecurityGroup
- DescribeAvailabilityZones
- DescribeCluster
- DescribeImages
- DescribeInstanceTypeOfferings
- DescribeInstances
- DescribeKeyPairs
- DescribeSecurityGroups
- DescribeSubnets
- DescribeVpcs
- DetachRolePolicy
- GetBootstrapBrokers
- GetCallerIdentity
- GetInstanceProfile
- GetPolicy
- GetRole
- RemoveRoleFromInstanceProfile
- RevokeSecurityGroupEgress
- RevokeSecurityGroupIngress
- RunInstances
- TerminateInstances
- Wait

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with push notifications

The following code example shows how to:

- Create an application
- Enable push notification channels
- Send a push notification
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS End User Messaging Push Getting Started Script
# This script creates an AWS End User Messaging Push application and demonstrates
# how to enable push notification channels and send a test message.
#
# Prerequisites:
# - AWS CLI installed and configured
# - Appropriate IAM permissions for Pinpoint operations
#
# Usage: ./aws-end-user-messaging-gs.sh [--auto-cleanup]

set -uo pipefail

# Security: Set secure umask for created files
umask 0077

# Set up logging
LOG_DIR="./aws-eump-logs"
mkdir -p "$LOG_DIR"
LOG_FILE="$LOG_DIR/aws-end-user-messaging-push-script-$(date +%Y%m%d-%H%M%S).log"
```

```
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting AWS End User Messaging Push setup script..."
echo "Logging to $LOG_FILE"
echo "Timestamp: $(date)"

# Security: Track created resources for cleanup
declare -a TEMP_FILES=()
declare -a AWS_RESOURCES=()

# Cleanup function with improved security
cleanup() {
    local exit_code=$?
    echo "Cleaning up temporary resources..."

    # Remove temporary files securely
    for temp_file in "${TEMP_FILES[@]+"${TEMP_FILES[@]}"}"; do
        if [ -f "$temp_file" ]; then
            shred -vfz -n 3 "$temp_file" 2>/dev/null || rm -f "$temp_file"
        fi
    done

    # Optionally delete AWS resources
    if [ "${DELETE_AWS_RESOURCES:-false}" = "true" ]; then
        for resource in "${AWS_RESOURCES[@]+"${AWS_RESOURCES[@]}"}"; do
            echo "Deleting AWS resource: $resource"
            aws pinpoint delete-app --application-id "$resource" 2>/dev/null || \
                echo "Warning: Failed to delete application $resource"
        done
    fi

    exit "$exit_code"
}

trap cleanup EXIT INT TERM

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2
    local ignore_error=${3:-false}

    if echo "$output" | grep -qi "error\|exception\|fail"; then
        echo "ERROR: Command failed: $cmd" >&2
```

```
        echo "Error details: $output" >&2

        if [ "$ignore_error" = "true" ]; then
            echo "Ignoring error and continuing..." >&2
            return 1
        else
            return 2
        fi
    fi

    return 0
}

# Function to validate AWS CLI is configured
validate_aws_cli() {
    echo "Validating AWS CLI configuration..."

    # Check if AWS CLI is installed
    if ! command -v aws &> /dev/null; then
        echo "ERROR: AWS CLI is not installed. Please install it first." >&2
        echo "Visit: https://docs.aws.amazon.com/cli/latest/userguide/getting-
started-install.html" >&2
        return 1
    fi

    # Check AWS CLI version
    AWS_VERSION=$(aws --version 2>&1 | head -n1)
    echo "AWS CLI version: $AWS_VERSION"

    # Verify credentials are set (check for credential env vars or config file)
    if ! aws sts get-caller-identity &> /dev/null; then
        echo "ERROR: AWS CLI credentials are not configured or invalid." >&2
        echo "Please configure credentials via environment variables, credential
 file, or 'aws configure'" >&2
        return 1
    fi

    # Get current AWS identity and region
    CALLER_IDENTITY=$(aws sts get-caller-identity)
    CURRENT_REGION=$(aws configure get region 2>/dev/null || echo "us-east-1")
    echo "AWS CLI configured for:"
    echo "$CALLER_IDENTITY"
    echo "Current region: $CURRENT_REGION"
    echo ""
```

```
    return 0
}

# Function to check if jq is available for JSON parsing
check_json_tools() {
    if command -v jq &> /dev/null; then
        USE_JQ=true
        echo "jq is available for JSON parsing"
    else
        USE_JQ=false
        echo "jq is not available, using grep for JSON parsing"
        echo "Consider installing jq for better JSON handling: https://
stedolan.github.io/jq/"
    fi
}

# Function to extract JSON values safely
extract_json_value() {
    local json=$1
    local key=$2

    if [ "$USE_JQ" = "true" ]; then
        echo "$json" | jq -r ".ApplicationResponse.$key // empty" 2>/dev/null ||
 echo ""
    else
        # Fallback to grep method with better validation
        echo "$json" | grep -o "\"$key\": \"[^\"]*" | cut -d'"' -f4 | head -n1 ||
 echo ""
    fi
}

# Function to validate required IAM permissions
validate_permissions() {
    echo "Validating IAM permissions..."

    # Test basic Pinpoint permissions
    if ! aws pinpoint get-apps > /dev/null 2>&1; then
        echo "WARNING: Unable to list Pinpoint applications." >&2
        echo "Please ensure you have appropriate IAM permissions for Pinpoint
 operations." >&2
        echo "Required permissions:" >&2
        echo "  - mobiletargeting:GetApps" >&2
        echo "  - mobiletargeting:CreateApp" >&2
```

```
        echo "  - mobiletargeting:DeleteApp" >&2
        echo "  - mobiletargeting:UpdateGcmChannel" >&2
        echo "  - mobiletargeting:UpdateApnsChannel" >&2
        echo "  - mobiletargeting:SendMessages" >&2
    else
        echo "Basic Pinpoint permissions validated."
    fi
}

# Function to validate input parameters
validate_input() {
    local app_name=$1

    # Validate app name length and characters
    if [ ${#app_name} -gt 64 ]; then
        echo "ERROR: Application name exceeds maximum length of 64 characters"
 >&2
        return 1
    fi

    if ! [[ "$app_name" =~ ^[a-zA-Z0-9_-]+$ ]]; then
        echo "ERROR: Application name contains invalid characters" >&2
        return 1
    fi

    return 0
}

# Function to create secure temporary files
create_temp_file() {
    local temp_file
    temp_file=$(mktemp) || {
        echo "ERROR: Failed to create temporary file" >&2
        return 1
    }
    chmod 600 "$temp_file"
    TEMP_FILES+=("$temp_file")
    echo "$temp_file"
}

# Validate prerequisites
if ! validate_aws_cli; then
    exit 1
fi
```

```
check_json_tools
validate_permissions

# Generate a random suffix for resource names to avoid conflicts
RANDOM_SUFFIX=$(LC_ALL=C tr -dc 'a-z0-9' < /dev/urandom | fold -w 8 | head -n1)
APP_NAME="PushNotificationApp-${RANDOM_SUFFIX}"

# Validate input
if ! validate_input "$APP_NAME"; then
    exit 1
fi

echo "Creating application with name: $APP_NAME"

# Step 1: Create an application
echo "Executing: aws pinpoint create-app --create-application-request Name=
${APP_NAME}"
CREATE_APP_OUTPUT=$(aws pinpoint create-app --create-application-request "Name=
${APP_NAME},tags={project=doc-smith,tutorial=aws-end-user-messaging-gs}" 2>&1)

if ! check_error "$CREATE_APP_OUTPUT" "create-app"; then
    exit 1
fi

echo "Application created successfully:"
echo "$CREATE_APP_OUTPUT"

# Extract the application ID from the output
APP_ID=$(extract_json_value "$CREATE_APP_OUTPUT" "Id")

if [ -z "$APP_ID" ] || [ "$APP_ID" = "null" ]; then
    echo "ERROR: Failed to extract application ID from output" >&2
    echo "Output was: $CREATE_APP_OUTPUT" >&2
    exit 1
fi

echo "Application ID: $APP_ID"
AWS_RESOURCES+=("$APP_ID")

# Step 2: Enable FCM (GCM) channel with a sample API key
echo ""
echo "========================================="
echo "ENABLING FCM (GCM) CHANNEL"
```

```
echo "========================================="
echo "Note: This is using a placeholder API key for demonstration purposes only."
echo "In a production environment, you should use your actual FCM API key from
 Firebase Console."
echo ""
echo "IMPORTANT: The following command will likely fail because we're using a
 placeholder API key."
echo "This is expected behavior for this demonstration script."

echo "Executing: aws pinpoint update-gcm-channel --application-id $APP_ID --gcm-
channel-request ..."
UPDATE_GCM_OUTPUT=$(aws pinpoint update-gcm-channel \
    --application-id "$APP_ID" \
    --gcm-channel-request '{"Enabled": true, "ApiKey": "sample-fcm-api-key-for-
demo-only"}' 2>&1)

# We'll ignore this specific error since we're using a placeholder API key
if check_error "$UPDATE_GCM_OUTPUT" "update-gcm-channel" "true"; then
    echo "FCM channel enabled successfully:"
    echo "$UPDATE_GCM_OUTPUT"
else
    echo "As expected, FCM channel update failed with the placeholder API key."
    echo "Error details: $UPDATE_GCM_OUTPUT"
    echo ""
    echo "To enable FCM in production:"
    echo "1. Go to Firebase Console (https://console.firebase.google.com/)"
    echo "2. Create or select your project"
    echo "3. Go to Project Settings > Cloud Messaging"
    echo "4. Copy the Server Key"
    echo "5. Replace 'sample-fcm-api-key-for-demo-only' with your actual Server
 Key"
fi

# Step 3: Try to enable APNS channel (this will also fail without real
 certificates)
echo ""
echo "========================================="
echo "ENABLING APNS CHANNEL (OPTIONAL)"
echo "========================================="
echo "Attempting to enable APNS channel with placeholder certificate..."
echo "This will also fail without real APNS certificates, which is expected."

# Create a placeholder APNS configuration
```

```
echo "Executing: aws pinpoint update-apns-channel --application-id $APP_ID --
apns-channel-request ..."
UPDATE_APNS_OUTPUT=$(aws pinpoint update-apns-channel \
    --application-id "$APP_ID" \
    --apns-channel-request '{"Enabled": true, "Certificate": "placeholder-
certificate", "PrivateKey": "placeholder-private-key"}' 2>&1)

if check_error "$UPDATE_APNS_OUTPUT" "update-apns-channel" "true"; then
    echo "APNS channel enabled successfully:"
    echo "$UPDATE_APNS_OUTPUT"
else
    echo "As expected, APNS channel update failed with placeholder certificates."
    echo "Error details: $UPDATE_APNS_OUTPUT"
    echo ""
    echo "To enable APNS in production:"
    echo "1. Generate APNS certificates from Apple Developer Console"
    echo "2. Convert certificates to PEM format"
    echo "3. Use the actual certificate and private key in the update-apns-
channel command"
fi

# Step 4: Create message files for different platforms
echo ""
echo "=========================================="
echo "CREATING MESSAGE FILES"
echo "=========================================="

# Create FCM message file securely
GCM_MESSAGE_FILE=$(create_temp_file)
echo "Creating FCM message file..."
cat > "$GCM_MESSAGE_FILE" << 'EOF'
{
  "Addresses": {
    "SAMPLE-DEVICE-TOKEN-FCM": {
      "ChannelType": "GCM"
    }
  },
  "MessageConfiguration": {
    "GCMMessage": {
      "Action": "OPEN_APP",
      "Body": "Hello from AWS End User Messaging Push! This is an FCM
 notification.",
      "Priority": "normal",
      "SilentPush": false,
```

```
      "Title": "My First FCM Push Notification",
      "TimeToLive": 30,
      "Data": {
        "key1": "value1",
        "key2": "value2"
      }
    }
  }
}
EOF

# Create APNS message file securely
APNS_MESSAGE_FILE=$(create_temp_file)
echo "Creating APNS message file..."
cat > "$APNS_MESSAGE_FILE" << 'EOF'
{
  "Addresses": {
    "SAMPLE-DEVICE-TOKEN-APNS": {
      "ChannelType": "APNS"
    }
  },
  "MessageConfiguration": {
    "APNSMessage": {
      "Action": "OPEN_APP",
      "Body": "Hello from AWS End User Messaging Push! This is an APNS
 notification.",
      "Priority": "normal",
      "SilentPush": false,
      "Title": "My First APNS Push Notification",
      "TimeToLive": 30,
      "Badge": 1,
      "Sound": "default"
    }
  }
}
EOF

echo "Message files created:"
echo "- FCM message file (for FCM/Android)"
echo "- APNS message file (for APNS/iOS)"
echo ""
echo "Note: These messages use placeholder device tokens and will not actually be
 delivered."
```

```
echo "To send real messages, you would need to replace the sample device tokens
 with actual ones."

# Step 5: Demonstrate how to send messages (this will fail with placeholder
 tokens)
echo ""
echo "========================================="
echo "DEMONSTRATING MESSAGE SENDING"
echo "========================================="
echo "Attempting to send FCM message (will fail with placeholder token)..."

echo "Executing: aws pinpoint send-messages --application-id $APP_ID --message-
request file://<gcm-message>"
SEND_FCM_OUTPUT=$(aws pinpoint send-messages \
    --application-id "$APP_ID" \
    --message-request "file://$GCM_MESSAGE_FILE" 2>&1)

if check_error "$SEND_FCM_OUTPUT" "send-messages (FCM)" "true"; then
    echo "FCM message sent successfully:"
    echo "$SEND_FCM_OUTPUT"
else
    echo "As expected, FCM message sending failed with placeholder token."
    echo "Error details: $SEND_FCM_OUTPUT"
fi

echo ""
echo "Attempting to send APNS message (will fail with placeholder token)..."

echo "Executing: aws pinpoint send-messages --application-id $APP_ID --message-
request file://<apns-message>"
SEND_APNS_OUTPUT=$(aws pinpoint send-messages \
    --application-id "$APP_ID" \
    --message-request "file://$APNS_MESSAGE_FILE" 2>&1)

if check_error "$SEND_APNS_OUTPUT" "send-messages (APNS)" "true"; then
    echo "APNS message sent successfully:"
    echo "$SEND_APNS_OUTPUT"
else
    echo "As expected, APNS message sending failed with placeholder token."
    echo "Error details: $SEND_APNS_OUTPUT"
fi

# Step 6: Show application details
echo ""
```

```
echo "============================================"
echo "APPLICATION DETAILS"
echo "============================================"
echo "Retrieving application details..."

echo "Executing: aws pinpoint get-app --application-id $APP_ID"
GET_APP_OUTPUT=$(aws pinpoint get-app --application-id "$APP_ID" 2>&1)
if check_error "$GET_APP_OUTPUT" "get-app"; then
    echo "Application details:"
    echo "$GET_APP_OUTPUT"
fi

# Display summary of created resources
echo ""
echo "============================================"
echo "RESOURCES CREATED"
echo "============================================"
echo "AWS Resources:"
for resource in "${AWS_RESOURCES[@]+"${AWS_RESOURCES[@]}"}"; do
    echo "- Application: $resource"
done

echo ""
echo "Files created:"
echo "- $LOG_FILE (script log)"

# Auto-cleanup information
echo ""
echo "============================================"
echo "CLEANUP INFORMATION"
echo "============================================"
echo "This script created AWS resources that may incur charges."
echo "AWS resources will be automatically cleaned up on script exit."
echo ""
echo "To manually delete resources later, use:"
echo "  aws pinpoint delete-app --application-id $APP_ID"

# Set flag to delete AWS resources on cleanup
DELETE_AWS_RESOURCES=true

echo ""
echo "============================================"
echo "SCRIPT COMPLETED SUCCESSFULLY"
echo "============================================"
```

```
echo "This script demonstrated:"
echo "1. Creating an AWS End User Messaging Push application"
echo "2. Attempting to enable FCM and APNS channels (with placeholder
 credentials)"
echo "3. Creating message templates for different platforms"
echo "4. Demonstrating message sending commands (with placeholder tokens)"
echo "5. Retrieving application details"
echo "6. Proper cleanup of resources"
echo ""
echo "Security best practices implemented:"
echo "- Secure temporary file handling with restricted permissions"
echo "- Input validation for application names"
echo "- Error handling with proper exit codes"
echo "- Credential validation before AWS operations"
echo "- Automatic cleanup on script exit"
echo "- Secure file destruction for sensitive data"
echo ""
echo "For production use:"
echo "- Replace placeholder API keys with real FCM server keys"
echo "- Replace placeholder certificates with real APNS certificates"
echo "- Replace placeholder device tokens with real device tokens"
echo "- Use AWS IAM roles instead of long-term credentials"
echo "- Implement comprehensive error handling"
echo "- Store sensitive credentials in AWS Secrets Manager or Parameter Store"
echo "- Enable CloudTrail for audit logging"
echo "- Use VPC endpoints for private AWS API access"
echo ""
echo "Log file: $LOG_FILE"
echo "Script completed at: $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - CreateApp
  - DeleteApp
  - GetApp
  - GetApps
  - GetCallerIdentity
  - SendMessages
  - UpdateApnsChannel
  - UpdateGcmChannel

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with search and analytics engines

The following code example shows how to:

- Create an OpenSearch Service domain
- Upload data to your domain
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon OpenSearch Service Getting Started Script - Version 8 Fixed
# This script creates an OpenSearch domain, uploads data, searches documents, and
 cleans up resources
# Based on the tested and working 4-tutorial-final.md

# SECURITY IMPROVEMENTS IN THIS VERSION:
# 1. Removed hardcoded passwords - now generated securely
# 2. Improved access policy to use principle of least privilege
# 3. Added credential masking in logs
# 4. Removed credentials from command output
# 5. Added input validation for AWS region
# 6. Improved error handling and validation
# 7. Added secure temporary file handling
# 8. Removed unnecessary curl insecurity flags
# 9. Added better secret management practices
# 10. Improved resource tagging for better governance
```

```
set -e  # Exit on any error

# Set up logging with restricted permissions
LOG_FILE="opensearch_tutorial_v8_fixed.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon OpenSearch Service tutorial script v8-fixed at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Track if domain was successfully created
DOMAIN_CREATED=false
DOMAIN_ACTIVE=false

# Secure temporary directory
TEMP_DIR=$(mktemp -d)
trap 'rm -rf "$TEMP_DIR"' EXIT
chmod 700 "$TEMP_DIR"

# Error handling function
handle_error() {
    echo "ERROR: $1"
    echo "Attempting to clean up resources..."
    cleanup_resources
    exit 1
}

# Function to clean up resources
cleanup_resources() {
    echo "Cleaning up resources..."

    if [[ "$DOMAIN_CREATED" == "true" ]]; then
        echo "Checking if domain $DOMAIN_NAME exists before attempting to
 delete..."

        # Check if domain exists before trying to delete
        if aws opensearch describe-domain --domain-name "$DOMAIN_NAME" > /dev/
null 2>&1; then
            echo "Domain $DOMAIN_NAME exists. Proceeding with deletion."
            aws opensearch delete-domain --domain-name "$DOMAIN_NAME"
            echo "Domain deletion initiated. This may take several minutes to
 complete."
```

```
        else
            echo "Domain $DOMAIN_NAME does not exist or is not accessible. No
 deletion needed."
        fi
    else
        echo "No domain was successfully created. Nothing to clean up."
    fi

    # Securely clean up sensitive files
    if [[ -d "$TEMP_DIR" ]]; then
        shred -vfz -n 3 "$TEMP_DIR"/* 2>/dev/null || true
        rm -rf "$TEMP_DIR"
    fi
}

# Generate a random identifier for resource names to avoid conflicts
RANDOM_ID=$(openssl rand -hex 4)
DOMAIN_NAME="movies-${RANDOM_ID}"
MASTER_USER="master-user"

# Generate secure password using openssl instead of hardcoding
MASTER_PASSWORD=$(openssl rand -base64 32 | tr -d "=+/" | cut -c1-25)

# Mask password in logs
echo "Using domain name: $DOMAIN_NAME"
echo "Using master username: $MASTER_USER"
echo "Using master password: [REDACTED]"

# Get AWS account ID (matches tutorial)
echo "Retrieving AWS account ID..."
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
if [[ $? -ne 0 ]] || [[ -z "$ACCOUNT_ID" ]]; then
    handle_error "Failed to retrieve AWS account ID. Please check your AWS
 credentials."
fi
echo "AWS Account ID: $ACCOUNT_ID"

# Get current region (matches tutorial)
echo "Retrieving current AWS region..."
AWS_REGION=$(aws configure get region)
if [[ -z "$AWS_REGION" ]]; then
    AWS_REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $AWS_REGION"
else
```

```
    echo "Using AWS region: $AWS_REGION"
fi

# Validate AWS region format
if ! [[ "$AWS_REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]$ ]]; then
    handle_error "Invalid AWS region format: $AWS_REGION"
fi

# Step 1: Create an OpenSearch Service Domain
echo "Creating OpenSearch Service domain..."
echo "This may take 15-30 minutes to complete."

# SECURITY IMPROVED: Use least privilege access policy
# This policy restricts access to specific actions and should be further
 restricted to specific principals in production
# Create the domain using CloudFormation (handles the 15-20 min creation wait)
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
STACK_NAME="tutorial-opensearch-${RANDOM_ID}"

echo "Creating domain $DOMAIN_NAME via CloudFormation..."
echo "This typically takes 15-20 minutes. CloudFormation will wait for
 completion."
aws cloudformation deploy \
  --template-file "$SCRIPT_DIR/cfn-opensearch-domain.yaml" \
  --stack-name "$STACK_NAME" \
  --parameter-overrides \
    DomainName="$DOMAIN_NAME" \
    MasterUserName="$MASTER_USER" \
    MasterUserPassword="$MASTER_PASSWORD" \
  --tags project=doc-smith tutorial=opensearch-service-gs \
  --no-fail-on-empty-changeset 2>&1 || handle_error "CloudFormation stack
 creation failed"

DOMAIN_CREATED=true
echo "Domain created successfully via CloudFormation."

# Get domain endpoint from stack outputs
DOMAIN_ENDPOINT=$(aws cloudformation describe-stacks \
  --stack-name "$STACK_NAME" \
  --query 'Stacks[0].Outputs[?OutputKey==`DomainEndpoint`].OutputValue' \
  --output text)

if [[ -z "$DOMAIN_ENDPOINT" ]] || [[ "$DOMAIN_ENDPOINT" == "None" ]]; then
    handle_error "Failed to get domain endpoint from CloudFormation outputs"
```

```
fi

echo "Domain endpoint: $DOMAIN_ENDPOINT"

# Wait for fine-grained access control to be fully ready
echo "Waiting for fine-grained access control to initialize..."
sleep 120

# Verify variables are set correctly (matches tutorial)
echo "Verifying configuration..."
echo "Domain endpoint: $DOMAIN_ENDPOINT"
echo "Master user: $MASTER_USER"
echo "Password set: $(if [ -n "$MASTER_PASSWORD" ]; then echo "Yes"; else echo
 "No"; fi)"

# Step 2: Upload Data to the Domain
echo "Preparing to upload data to the domain..."

# Create a file for the single document (matches tutorial exactly)
echo "Creating single document JSON file..."
SINGLE_MOVIE_FILE="$TEMP_DIR/single_movie.json"
cat > "$SINGLE_MOVIE_FILE" << 'EOF'
{
  "director": "Burton, Tim",
  "genre": ["Comedy","Sci-Fi"],
  "year": 1996,
  "actor": ["Jack Nicholson","Pierce Brosnan","Sarah Jessica Parker"],
  "title": "Mars Attacks!"
}
EOF
chmod 600 "$SINGLE_MOVIE_FILE"

# Create a file for bulk documents (matches tutorial exactly)
echo "Creating bulk documents JSON file..."
BULK_MOVIES_FILE="$TEMP_DIR/bulk_movies.json"
cat > "$BULK_MOVIES_FILE" << 'EOF'
{ "index" : { "_index": "movies", "_id" : "2" } }
{"director": "Frankenheimer, John", "genre": ["Drama", "Mystery", "Thriller",
 "Crime"], "year": 1962, "actor": ["Lansbury, Angela", "Sinatra, Frank", "Leigh,
 Janet", "Harvey, Laurence", "Silva, Henry", "Frees, Paul", "Gregory, James",
 "Bissell, Whit", "McGiver, John", "Parrish, Leslie", "Edwards, James", "Flowers,
 Bess", "Dhiegh, Khigh", "Payne, Julie", "Kleeb, Helen", "Gray, Joe", "Nalder,
 Reggie", "Stevens, Bert", "Masters, Michael", "Lowell, Tom"], "title": "The
 Manchurian Candidate"}
```

```
{ "index" : { "_index": "movies", "_id" : "3" } }
{"director": "Baird, Stuart", "genre": ["Action", "Crime", "Thriller"], "year":
 1998, "actor": ["Downey Jr., Robert", "Jones, Tommy Lee", "Snipes, Wesley",
 "Pantoliano, Joe", "Jacob, Irène", "Nelligan, Kate", "Roebuck, Daniel",
 "Malahide, Patrick", "Richardson, LaTanya", "Wood, Tom", "Kosik, Thomas",
 "Stellate, Nick", "Minkoff, Robert", "Brown, Spitfire", "Foster, Reese",
 "Spielbauer, Bruce", "Mukherji, Kevin", "Cray, Ed", "Fordham, David", "Jett,
 Charlie"], "title": "U.S. Marshals"}
{ "index" : { "_index": "movies", "_id" : "4" } }
{"director": "Ray, Nicholas", "genre": ["Drama", "Romance"], "year": 1955,
 "actor": ["Hopper, Dennis", "Wood, Natalie", "Dean, James", "Mineo, Sal",
 "Backus, Jim", "Platt, Edward", "Ray, Nicholas", "Hopper, William", "Allen,
 Corey", "Birch, Paul", "Hudson, Rochelle", "Doran, Ann", "Hicks, Chuck", "Leigh,
 Nelson", "Williams, Robert", "Wessel, Dick", "Bryar, Paul", "Sessions, Almira",
 "McMahon, David", "Peters Jr., House"], "title": "Rebel Without a Cause"}
EOF
chmod 600 "$BULK_MOVIES_FILE"

# Check if curl is installed
if ! command -v curl &> /dev/null; then
    echo "Warning: curl is not installed. Skipping data upload and search steps."
    echo "You can manually upload the data later using the commands in the
 tutorial."
else
    # IMPROVED: Test authentication with multiple approaches
    echo "Testing authentication with the OpenSearch domain..."
    echo "This test checks if fine-grained access control is ready for data
 operations."

    # Create credentials file for secure handling
    CREDENTIALS_FILE="$TEMP_DIR/.credentials"
    echo "$MASTER_USER:$MASTER_PASSWORD" > "$CREDENTIALS_FILE"
    chmod 600 "$CREDENTIALS_FILE"

    # Test 1: Basic authentication with root endpoint
    echo "Testing basic authentication with root endpoint..."
    AUTH_TEST_RESULT=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
        --netrc-file "$CREDENTIALS_FILE" \
        --request GET \
        "https://${DOMAIN_ENDPOINT}/" 2>&1)

    echo "Basic auth test result:"
    echo "$AUTH_TEST_RESULT"
```

```
    # Extract HTTP status code
    HTTP_CODE=$(echo "$AUTH_TEST_RESULT" | grep "HTTP_CODE:" | cut -d: -f2)

    # Function to check if HTTP code is 2xx
    is_success_code() {
        local code=$1
        if [[ "$code" =~ ^2[0-9][0-9]$ ]]; then
            return 0
        else
            return 1
        fi
    }

    # Check if basic authentication test was successful (200 or 2xx status codes)
    if is_success_code "$HTTP_CODE"; then
        echo "# Basic authentication test successful! (HTTP $HTTP_CODE)"
        AUTH_SUCCESS=true
        AUTH_METHOD="basic"
    else
        echo "Basic authentication failed with HTTP code: $HTTP_CODE"

        # Test 2: Try cluster health endpoint
        echo "Testing with cluster health endpoint..."
        HEALTH_TEST_RESULT=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
            --netrc-file "$CREDENTIALS_FILE" \
            --request GET \
            "https://${DOMAIN_ENDPOINT}/_cluster/health" 2>&1)

        echo "Cluster health test result:"
        echo "$HEALTH_TEST_RESULT"

        HEALTH_HTTP_CODE=$(echo "$HEALTH_TEST_RESULT" | grep "HTTP_CODE:" | cut -
d: -f2)

        if is_success_code "$HEALTH_HTTP_CODE"; then
            echo "# Cluster health authentication test successful! (HTTP
 $HEALTH_HTTP_CODE)"
            AUTH_SUCCESS=true
            AUTH_METHOD="basic"
        else
            echo "Cluster health authentication also failed with HTTP code:
 $HEALTH_HTTP_CODE"

            # Check for specific error patterns
```

```
            if echo "$AUTH_TEST_RESULT" | grep -q "anonymous is not authorized";
 then
                echo "Error: Request is being treated as anonymous
 (authentication not working)"
            elif echo "$AUTH_TEST_RESULT" | grep -q "Unauthorized"; then
                echo "Error: Authentication credentials rejected"
            elif echo "$AUTH_TEST_RESULT" | grep -q "Forbidden"; then
                echo "Error: Authentication succeeded but access is forbidden"
            fi

            echo "Waiting additional time and retrying with exponential
 backoff..."

            # Retry authentication test with exponential backoff
            AUTH_RETRY_COUNT=0
            MAX_AUTH_RETRIES=5
            WAIT_TIME=60
            AUTH_SUCCESS=false

            while [[ $AUTH_RETRY_COUNT -lt $MAX_AUTH_RETRIES ]]; do
                echo "Retrying authentication test (attempt $((AUTH_RETRY_COUNT
+1))/$MAX_AUTH_RETRIES) after ${WAIT_TIME} seconds..."
                sleep $WAIT_TIME

                # Try both endpoints
                AUTH_TEST_RESULT=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
                    --netrc-file "$CREDENTIALS_FILE" \
                    --request GET \
                    "https://${DOMAIN_ENDPOINT}/" 2>&1)

                HTTP_CODE=$(echo "$AUTH_TEST_RESULT" | grep "HTTP_CODE:" | cut -
d: -f2)

                echo "Retry result (HTTP $HTTP_CODE):"
                echo "$AUTH_TEST_RESULT"

                if is_success_code "$HTTP_CODE"; then
                    echo "# Authentication test successful after retry! (HTTP
 $HTTP_CODE)"
                    AUTH_SUCCESS=true
                    AUTH_METHOD="basic"
                    break
                fi
```

```
                # Also try cluster health
                HEALTH_TEST_RESULT=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
                    --netrc-file "$CREDENTIALS_FILE" \
                    --request GET \
                    "https://${DOMAIN_ENDPOINT}/_cluster/health" 2>&1)

                HEALTH_HTTP_CODE=$(echo "$HEALTH_TEST_RESULT" | grep "HTTP_CODE:"
| cut -d: -f2)

                if is_success_code "$HEALTH_HTTP_CODE"; then
                    echo "# Cluster health authentication successful after retry!
(HTTP $HEALTH_HTTP_CODE)"
                    AUTH_SUCCESS=true
                    AUTH_METHOD="basic"
                    break
                fi

                AUTH_RETRY_COUNT=$((AUTH_RETRY_COUNT+1))
                # Exponential backoff: double the wait time each retry (max 10
minutes)
                WAIT_TIME=$((WAIT_TIME * 2))
                if [[ $WAIT_TIME -gt 600 ]]; then
                    WAIT_TIME=600
                fi
            done
        fi
    fi

    # Proceed with data operations if authentication is working
    if [[ "$AUTH_SUCCESS" == "true" ]]; then
        echo "Authentication successful using $AUTH_METHOD method. Proceeding
with data operations."

        # Upload single document (matches tutorial exactly)
        echo "Uploading single document..."
        UPLOAD_RESULT=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
            --netrc-file "$CREDENTIALS_FILE" \
            --request PUT \
            --header 'Content-Type: application/json' \
            --data @"$SINGLE_MOVIE_FILE" \
            "https://${DOMAIN_ENDPOINT}/movies/_doc/1" 2>&1)

        echo "Upload response:"
        echo "$UPLOAD_RESULT"
```

```
        UPLOAD_HTTP_CODE=$(echo "$UPLOAD_RESULT" | grep "HTTP_CODE:" | cut -d: -
f2)
        if is_success_code "$UPLOAD_HTTP_CODE" && echo "$UPLOAD_RESULT" | grep -q
 '"result"'; then
            echo "# Single document uploaded successfully! (HTTP
 $UPLOAD_HTTP_CODE)"
        else
            echo "# Warning: Single document upload may have failed (HTTP
 $UPLOAD_HTTP_CODE)"
        fi

        # Upload bulk documents (matches tutorial exactly)
        echo "Uploading bulk documents..."
        BULK_RESULT=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
            --netrc-file "$CREDENTIALS_FILE" \
            --request POST \
            --header 'Content-Type: application/x-ndjson' \
            --data-binary @"$BULK_MOVIES_FILE" \
            "https://${DOMAIN_ENDPOINT}/movies/_bulk" 2>&1)

        echo "Bulk upload response:"
        echo "$BULK_RESULT"

        BULK_HTTP_CODE=$(echo "$BULK_RESULT" | grep "HTTP_CODE:" | cut -d: -f2)
        if is_success_code "$BULK_HTTP_CODE" && echo "$BULK_RESULT" | grep -q
 '"errors": false'; then
            echo "# Bulk documents uploaded successfully! (HTTP $BULK_HTTP_CODE)"
        else
            echo "# Warning: Bulk document upload may have failed (HTTP
 $BULK_HTTP_CODE)"
        fi

        # Wait a moment for indexing
        echo "Waiting for documents to be indexed..."
        sleep 5

        # Step 3: Search Documents (matches tutorial exactly)
        echo "Searching for documents containing 'mars'..."
        SEARCH_RESULT=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
            --netrc-file "$CREDENTIALS_FILE" \
            --request GET \
            "https://${DOMAIN_ENDPOINT}/movies/_search?q=mars&pretty=true" 2>&1)
```

```
        SEARCH_HTTP_CODE=$(echo "$SEARCH_RESULT" | grep "HTTP_CODE:" | cut -d: -
f2)
        echo "Search results for 'mars' (HTTP $SEARCH_HTTP_CODE):"
        echo "$SEARCH_RESULT"

        echo "Searching for documents containing 'rebel'..."
        REBEL_SEARCH=$(curl -s -w "\nHTTP_CODE:%{http_code}" \
            --netrc-file "$CREDENTIALS_FILE" \
            --request GET \
            "https://${DOMAIN_ENDPOINT}/movies/_search?q=rebel&pretty=true" 2>&1)

        REBEL_HTTP_CODE=$(echo "$REBEL_SEARCH" | grep "HTTP_CODE:" | cut -d: -f2)
        echo "Search results for 'rebel' (HTTP $REBEL_HTTP_CODE):"
        echo "$REBEL_SEARCH"

        # Verify search results
        if is_success_code "$SEARCH_HTTP_CODE" && echo "$SEARCH_RESULT" | grep -q
 '"hits"'; then
            echo "# Search functionality is working!"
        else
            echo "# Warning: Search may not be working properly."
        fi

    else
        echo ""
        echo "========================================="
        echo "AUTHENTICATION TROUBLESHOOTING"
        echo "========================================="
        echo "Authentication failed after all retries. This may be due to:"
        echo "1. Fine-grained access control not fully initialized (most common)"
        echo "2. Domain configuration issues"
        echo "3. Network connectivity issues"
        echo "4. AWS credentials or permissions issues"
        echo ""
        echo "DOMAIN CONFIGURATION DEBUG:"
        echo "Let's check the domain configuration..."

        # Debug domain configuration
        DOMAIN_CONFIG=$(aws opensearch describe-domain --domain-
name "$DOMAIN_NAME" --query 'DomainStatus.{AdvancedSecurityOptions:
 AdvancedSecurityOptions, AccessPolicies: AccessPolicies}' --output json 2>&1)
        echo "Domain configuration:"
        echo "$DOMAIN_CONFIG"
```

```
        echo ""
        echo "MANUAL TESTING COMMANDS:"
        echo "You can try these commands manually in 10-15 minutes:"
        echo ""
        echo "# Test basic authentication:"
        echo "curl --user \"${MASTER_USER}:[PASSWORD]\" \"https://
${DOMAIN_ENDPOINT}/\""
        echo ""
        echo "# Test cluster health:"
        echo "curl --user \"${MASTER_USER}:[PASSWORD]\" \"https://
${DOMAIN_ENDPOINT}/_cluster/health\""
        echo ""
        echo "# Upload single document:"
        echo "curl --user \"${MASTER_USER}:[PASSWORD]\" --request PUT --
header 'Content-Type: application/json' --data @single_movie.json \"https://
${DOMAIN_ENDPOINT}/movies/_doc/1\""
        echo ""
        echo "# Search for documents:"
        echo "curl --user \"${MASTER_USER}:[PASSWORD]\" \"https://
${DOMAIN_ENDPOINT}/movies/_search?q=mars&pretty=true\""
        echo ""
        echo "TROUBLESHOOTING TIPS:"
        echo "- Wait 10-15 more minutes and try the manual commands"
        echo "- Check AWS CloudTrail logs for authentication errors"
        echo "- Verify your AWS region is correct: $AWS_REGION"
        echo "- Ensure your AWS credentials have OpenSearch permissions"
        echo "- Try accessing OpenSearch Dashboards to verify the master user
 works"
        echo ""
        echo "Skipping data upload and search operations for now."
        echo "The domain is created and accessible via OpenSearch Dashboards."
    fi

    # Securely remove credentials file
    shred -vfz -n 3 "$CREDENTIALS_FILE" 2>/dev/null || true
fi

# Display OpenSearch Dashboards URL (matches tutorial)
echo ""
echo "========================================="
echo "OPENSEARCH DASHBOARDS ACCESS"
echo "========================================="
echo "OpenSearch Dashboards URL: https://${DOMAIN_ENDPOINT}/_dashboards/"
echo "Username: $MASTER_USER"
```

```
echo "Password: [REDACTED - stored securely]"
echo ""
echo "You can access OpenSearch Dashboards using the credentials provided during
 domain creation."
echo "If you uploaded data successfully, you can create an index pattern for
 'movies'."
echo ""

# Summary of created resources
echo ""
echo "========================================="
echo "RESOURCES CREATED"
echo "========================================="
echo "OpenSearch Domain Name: $DOMAIN_NAME"
echo "OpenSearch Domain Endpoint: $DOMAIN_ENDPOINT"
echo "AWS Region: $AWS_REGION"
echo "Master Username: $MASTER_USER"
echo ""
echo "ESTIMATED COST: ~$0.038/hour (~$0.91/day) until deleted"
echo ""
echo "Make sure to save the domain name and endpoint for future reference."
echo ""

# Auto-confirm cleanup: Yes, clean up all resources
echo ""
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Automatically cleaning up all created resources..."
CLEANUP_CHOICE="y"

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    echo "Cleaning up resources..."
    if [ -n "${STACK_NAME:-}" ]; then
        echo "Deleting CloudFormation stack $STACK_NAME..."
        aws cloudformation delete-stack --stack-name "$STACK_NAME"
        echo "# Stack deletion initiated. This may take several minutes."
        echo "  Monitor: aws cloudformation describe-stacks --stack-name
 $STACK_NAME"
    else
        aws opensearch delete-domain --domain-name "$DOMAIN_NAME" 2>/dev/null
        echo "# Domain deletion initiated."
    fi
else
```

```
    echo "Resources will NOT be deleted automatically."
    echo ""
    echo "To delete later:"
    echo "  aws cloudformation delete-stack --stack-name ${STACK_NAME:-tutorial-
opensearch-XXXX}"
    echo ""
    echo "# IMPORTANT: Keeping these resources will incur ongoing AWS charges!"
fi

# Clean up temporary files (handled by trap)
echo "Cleaning up temporary files..."

# Disable the trap since we're handling cleanup manually
trap - EXIT

# Final cleanup
rm -rf "$TEMP_DIR" 2>/dev/null || true

echo ""
echo "============================================="
echo "SCRIPT COMPLETED SUCCESSFULLY"
echo "============================================="
echo "Script completed at $(date)"
echo "All output has been logged to: $LOG_FILE"
echo ""
echo "Next steps:"
echo "1. Access OpenSearch Dashboards at: https://${DOMAIN_ENDPOINT}/
_dashboards/"
echo "2. Create visualizations and dashboards"
echo "3. Explore the OpenSearch API"
echo "4. Remember to delete resources when done to avoid charges"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - CreateDomain
  - DeleteDomain
  - DescribeDomain
  - GetCallerIdentity

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with web application firewalls

The following code example shows how to:

- Create a web ACL
- Add a string match rule
- Add managed rules
- Configure logging
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS WAF Getting Started Script
# This script creates a Web ACL with a string match rule and AWS Managed Rules,
# associates it with a CloudFront distribution, and then cleans up all resources.

set -euo pipefail

# Security: Restrict file permissions
umask 077

# Set up logging with secure file handling
LOG_FILE="waf-tutorial.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "================================================="
echo "AWS WAF Getting Started Tutorial"
echo "================================================="
```

```
echo "This script will create AWS WAF resources and associate"
echo "them with a CloudFront distribution."
echo ""

# Maximum number of retries for operations
MAX_RETRIES=3

# Function to handle errors securely
handle_error() {
    local error_msg="$1"
    echo "ERROR: $error_msg" >&2
    echo "Check the log file for details: $LOG_FILE" >&2
    cleanup_resources
    exit 1
}

# Function to validate AWS CLI JSON output
validate_json() {
    local json_string="$1"
    if ! echo "$json_string" | jq empty 2>/dev/null; then
        return 1
    fi
    return 0
}

# Function to safely extract JSON values
extract_json_value() {
    local json_string="$1"
    local key_path="$2"

    if ! validate_json "$json_string"; then
        return 1
    fi

    echo "$json_string" | jq -r "$key_path" 2>/dev/null || return 1
}

# Function to clean up resources securely
cleanup_resources() {
    echo ""
    echo "=================================================="
    echo "CLEANING UP RESOURCES"
    echo "=================================================="
```

```
    if [ -n "${DISTRIBUTION_ID:-}" ] && [ -n "${WEB_ACL_ARN:-}" ]; then
        echo "Disassociating Web ACL from CloudFront distribution..."
        local account_id
        account_id=$(aws sts get-caller-identity --query Account --output text
 2>/dev/null) || account_id=""

        if [ -z "$account_id" ]; then
            echo "Warning: Could not retrieve AWS account ID"
            return
        fi

        local disassociate_result
        disassociate_result=$(aws wafv2 disassociate-web-acl \
            --resource-arn "arn:aws:cloudfront::${account_id}:distribution/
${DISTRIBUTION_ID}" \
            --region us-east-1 2>&1) || true

        if echo "$disassociate_result" | grep -qi "error"; then
            echo "Warning: Failed to disassociate Web ACL: $disassociate_result"
        else
            echo "Web ACL disassociated successfully."
        fi
    fi

    if [ -n "${WEB_ACL_ID:-}" ] && [ -n "${WEB_ACL_NAME:-}" ]; then
        echo "Deleting Web ACL..."

        local get_result
        get_result=$(aws wafv2 get-web-acl \
            --name "$WEB_ACL_NAME" \
            --scope CLOUDFRONT \
            --id "$WEB_ACL_ID" \
            --region us-east-1 2>&1) || true

        if echo "$get_result" | grep -qi "error"; then
            echo "Warning: Failed to get Web ACL for deletion: $get_result"
            echo "You may need to manually delete the Web ACL using the AWS
 Console."
        else
            local latest_token
            latest_token=$(extract_json_value "$get_result" '.WebACL.LockToken'
 2>/dev/null) || latest_token=""

            if [ -n "$latest_token" ]; then
```

```
                local delete_result
                delete_result=$(aws wafv2 delete-web-acl \
                    --name "$WEB_ACL_NAME" \
                    --scope CLOUDFRONT \
                    --id "$WEB_ACL_ID" \
                    --lock-token "$latest_token" \
                    --region us-east-1 2>&1) || true

                if echo "$delete_result" | grep -qi "error"; then
                    echo "Warning: Failed to delete Web ACL: $delete_result"
                    echo "You may need to manually delete the Web ACL using the
 AWS Console."
                else
                    echo "Web ACL deleted successfully."
                fi
            else
                echo "Warning: Could not extract lock token for deletion. You may
 need to manually delete the Web ACL."
            fi
        fi
    fi

    echo "Cleanup process completed."
}

# Security: Trap EXIT to ensure cleanup on any exit
trap cleanup_resources EXIT

# Generate a random identifier for resource names using secure method
RANDOM_ID=$(openssl rand -hex 4) || handle_error "Failed to generate random ID"
WEB_ACL_NAME="MyWebACL-${RANDOM_ID}"
METRIC_NAME="MyWebACLMetrics-${RANDOM_ID}"

echo "Using Web ACL name: $WEB_ACL_NAME"

# Step 1: Create a Web ACL
echo ""
echo "================================================="
echo "STEP 1: Creating Web ACL"
echo "================================================="

local create_result
create_result=$(aws wafv2 create-web-acl \
    --name "$WEB_ACL_NAME" \
```

```
    --scope "CLOUDFRONT" \
    --default-action Allow={} \
    --visibility-config
 "SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=
$METRIC_NAME" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-waf-gs \
    --region us-east-1 2>&1) || handle_error "Failed to create Web ACL"

if ! validate_json "$create_result"; then
    handle_error "Invalid JSON response from create-web-acl"
fi

# Extract Web ACL ID, ARN, and Lock Token from the response
WEB_ACL_ID=$(extract_json_value "$create_result" '.Summary.Id') || handle_error
 "Failed to extract Web ACL ID"
WEB_ACL_ARN=$(extract_json_value "$create_result" '.Summary.ARN') || handle_error
 "Failed to extract Web ACL ARN"
LOCK_TOKEN=$(extract_json_value "$create_result" '.Summary.LockToken') ||
 handle_error "Failed to extract Lock Token"

echo "Web ACL created successfully with ID: $WEB_ACL_ID"
echo "Lock Token: [REDACTED]"

# Step 2: Add a String Match Rule
echo ""
echo "=================================================="
echo "STEP 2: Adding String Match Rule"
echo "=================================================="

for ((i=1; i<=MAX_RETRIES; i++)); do
    echo "Attempt $i to add string match rule..."

    # Get the latest lock token before updating
    local get_result
    get_result=$(aws wafv2 get-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope CLOUDFRONT \
        --id "$WEB_ACL_ID" \
        --region us-east-1 2>&1) || true

    if echo "$get_result" | grep -qi "error"; then
        echo "Warning: Failed to get Web ACL for update: $get_result"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to get Web ACL after $MAX_RETRIES attempts"
```

```
        fi
        sleep 2
        continue
    fi

    if ! validate_json "$get_result"; then
        echo "Warning: Invalid JSON response from get-web-acl"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Invalid JSON response after $MAX_RETRIES attempts"
        fi
        sleep 2
        continue
    fi

    local latest_token
    latest_token=$(extract_json_value "$get_result" '.WebACL.LockToken' 2>/dev/
null) || true

    if [ -z "$latest_token" ]; then
        echo "Warning: Could not extract lock token for update"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to extract lock token after $MAX_RETRIES
 attempts"
        fi
        sleep 2
        continue
    fi

    local update_result
    update_result=$(aws wafv2 update-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope "CLOUDFRONT" \
        --id "$WEB_ACL_ID" \
        --lock-token "$latest_token" \
        --default-action Allow={} \
        --rules '[{
            "Name": "UserAgentRule",
            "Priority": 0,
            "Statement": {
                "ByteMatchStatement": {
                    "SearchString": "MyAgent",
                    "FieldToMatch": {
                        "SingleHeader": {
                            "Name": "user-agent"
```

```
                        }
                    },
                    "TextTransformations": [
                        {
                            "Priority": 0,
                            "Type": "NONE"
                        }
                    ],
                    "PositionalConstraint": "EXACTLY"
                }
            },
            "Action": {
                "Count": {}
            },
            "VisibilityConfig": {
                "SampledRequestsEnabled": true,
                "CloudWatchMetricsEnabled": true,
                "MetricName": "UserAgentRuleMetric"
            }
        }]' \
        --visibility-config
 "SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=
$METRIC_NAME" \
        --region us-east-1 2>&1) || true

    if echo "$update_result" | grep -qi "WAFOptimisticLockException"; then
        echo "Optimistic lock exception encountered. Will retry with new lock
 token."
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to add string match rule after $MAX_RETRIES
 attempts"
        fi
        sleep 2
        continue
    elif echo "$update_result" | grep -qi "error"; then
        handle_error "Failed to add string match rule: $update_result"
    else
        echo "String match rule added successfully."
        break
    fi
done

# Step 3: Add AWS Managed Rules
echo ""
```

```
echo "=================================================="
echo "STEP 3: Adding AWS Managed Rules"
echo "=================================================="

for ((i=1; i<=MAX_RETRIES; i++)); do
    echo "Attempt $i to add AWS Managed Rules..."

    # Get the latest lock token before updating
    local get_result
    get_result=$(aws wafv2 get-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope CLOUDFRONT \
        --id "$WEB_ACL_ID" \
        --region us-east-1 2>&1) || true

    if echo "$get_result" | grep -qi "error"; then
        echo "Warning: Failed to get Web ACL for update: $get_result"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to get Web ACL after $MAX_RETRIES attempts"
        fi
        sleep 2
        continue
    fi

    if ! validate_json "$get_result"; then
        echo "Warning: Invalid JSON response from get-web-acl"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Invalid JSON response after $MAX_RETRIES attempts"
        fi
        sleep 2
        continue
    fi

    local latest_token
    latest_token=$(extract_json_value "$get_result" '.WebACL.LockToken' 2>/dev/
null) || true

    if [ -z "$latest_token" ]; then
        echo "Warning: Could not extract lock token for update"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to extract lock token after $MAX_RETRIES
 attempts"
        fi
        sleep 2
```

```
        continue
    fi

    local update_result
    update_result=$(aws wafv2 update-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope "CLOUDFRONT" \
        --id "$WEB_ACL_ID" \
        --lock-token "$latest_token" \
        --default-action Allow={} \
        --rules '[{
            "Name": "UserAgentRule",
            "Priority": 0,
            "Statement": {
                "ByteMatchStatement": {
                    "SearchString": "MyAgent",
                    "FieldToMatch": {
                        "SingleHeader": {
                            "Name": "user-agent"
                        }
                    },
                    "TextTransformations": [
                        {
                            "Priority": 0,
                            "Type": "NONE"
                        }
                    ],
                    "PositionalConstraint": "EXACTLY"
                }
            },
            "Action": {
                "Count": {}
            },
            "VisibilityConfig": {
                "SampledRequestsEnabled": true,
                "CloudWatchMetricsEnabled": true,
                "MetricName": "UserAgentRuleMetric"
            }
        },
        {
            "Name": "AWS-AWSManagedRulesCommonRuleSet",
            "Priority": 1,
            "Statement": {
                "ManagedRuleGroupStatement": {
```

```
                    "VendorName": "AWS",
                    "Name": "AWSManagedRulesCommonRuleSet",
                    "ExcludedRules": []
                }
            },
            "OverrideAction": {
                "Count": {}
            },
            "VisibilityConfig": {
                "SampledRequestsEnabled": true,
                "CloudWatchMetricsEnabled": true,
                "MetricName": "AWS-AWSManagedRulesCommonRuleSet"
            }
        }]' \
        --visibility-config
 "SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=
$METRIC_NAME" \
        --region us-east-1 2>&1) || true

    if echo "$update_result" | grep -qi "WAFOptimisticLockException"; then
        echo "Optimistic lock exception encountered. Will retry with new lock
 token."
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to add AWS Managed Rules after $MAX_RETRIES
 attempts"
        fi
        sleep 2
        continue
    elif echo "$update_result" | grep -qi "error"; then
        handle_error "Failed to add AWS Managed Rules: $update_result"
    else
        echo "AWS Managed Rules added successfully."
        break
    fi
done

# Step 4: List CloudFront distributions
echo ""
echo "=================================================="
echo "STEP 4: Listing CloudFront Distributions"
echo "=================================================="

local cf_result
```

```
cf_result=$(aws cloudfront list-distributions --query "DistributionList.Items[*].
{Id:Id,DomainName:DomainName}" --output table 2>&1) || cf_result=""

if echo "$cf_result" | grep -qi "error"; then
    echo "Warning: Failed to list CloudFront distributions: $cf_result"
    echo "Continuing without CloudFront association."
    DISTRIBUTION_ID=""
else
    echo "$cf_result"

    # Auto-select first CloudFront distribution if available
    echo ""
    echo "=================================================="
    echo "STEP 5: Associate Web ACL with CloudFront Distribution"
    echo "=================================================="

    local first_dist
    first_dist=$(aws cloudfront list-distributions --query
 "DistributionList.Items[0].Id" --output text 2>&1) || first_dist=""

    if [ -n "$first_dist" ] && [ "$first_dist" != "None" ] && ! echo
 "$first_dist" | grep -qi "error"; then
        DISTRIBUTION_ID="$first_dist"
        echo "Using CloudFront distribution: $DISTRIBUTION_ID"

        local account_id
        account_id=$(aws sts get-caller-identity --query Account --output text
 2>/dev/null) || account_id=""

        if [ -z "$account_id" ]; then
            echo "Warning: Could not retrieve AWS account ID for association"
            DISTRIBUTION_ID=""
        else
            local associate_result
            associate_result=$(aws wafv2 associate-web-acl \
                --web-acl-arn "$WEB_ACL_ARN" \
                --resource-arn "arn:aws:cloudfront::${account_id}:distribution/
${DISTRIBUTION_ID}" \
                --region us-east-1 2>&1) || true

            if echo "$associate_result" | grep -qi "error"; then
                echo "Warning: Failed to associate Web ACL with CloudFront
 distribution: $associate_result"
                echo "Continuing without CloudFront association."
```

```
                DISTRIBUTION_ID=""
            else
                echo "Web ACL associated with CloudFront distribution
 successfully."
            fi
        fi
    else
        echo "No CloudFront distributions available. Skipping association."
        DISTRIBUTION_ID=""
    fi
fi

# Display summary of created resources
echo ""
echo "=============================================="
echo "RESOURCE SUMMARY"
echo "=============================================="
echo "Web ACL Name: $WEB_ACL_NAME"
echo "Web ACL ID: $WEB_ACL_ID"
echo "Web ACL ARN: $WEB_ACL_ARN"
if [ -n "${DISTRIBUTION_ID:-}" ]; then
    echo "Associated CloudFront Distribution: $DISTRIBUTION_ID"
fi
echo ""

# Auto-confirm cleanup
echo "=============================================="
echo "CLEANUP CONFIRMATION"
echo "=============================================="
echo "Proceeding with automatic cleanup of all created resources..."

echo ""
echo "=============================================="
echo "Tutorial completed!"
echo "=============================================="
echo "Log file: $LOG_FILE"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AssociateWebAcl
  - CreateWebAcl
  - DeleteWebAcl
  - DisassociateWebAcl

- GetCallerIdentity
- GetWebAcl
- ListDistributions
- UpdateWebAcl

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Moving hardcoded secrets to secure secret storage

The following code example shows how to:

- Create IAM roles
- Create a secret in Secrets Manager
- Update your application code
- Update the secret
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Script to move hardcoded secrets to AWS Secrets Manager
# This script demonstrates how to create IAM roles, store a secret in AWS Secrets
 Manager,
# and set up appropriate permissions

set -euo pipefail
```

```
# Set up logging
LOG_FILE="secrets_manager_tutorial.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting AWS Secrets Manager tutorial script at $(date)"
echo "============================================================"

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2

    if echo "$output" | grep -qi "error"; then
        echo "ERROR: Command failed: $cmd"
        echo "$output"
        cleanup_resources
        exit 1
    fi
}

# Function to generate a random identifier using secure method
generate_random_id() {
    python3 -c "import secrets; print('sm' + secrets.token_hex(4))"
}

# Function to safely clean up resources
cleanup_resources() {
    echo ""
    echo "========================================="
    echo "RESOURCES CREATED"
    echo "========================================="

    if [ -n "${SECRET_NAME:-}" ]; then
        echo "Secret: $SECRET_NAME"
    fi

    if [ -n "${RUNTIME_ROLE_NAME:-}" ]; then
        echo "IAM Role: $RUNTIME_ROLE_NAME"
    fi

    if [ -n "${ADMIN_ROLE_NAME:-}" ]; then
        echo "IAM Role: $ADMIN_ROLE_NAME"
    fi
```

```
    echo ""
    echo "========================================="
    echo "CLEANUP CONFIRMATION"
    echo "========================================="
    echo "Cleaning up all created resources..."

    # Delete secret if it exists
    if [ -n "${SECRET_NAME:-}" ]; then
        echo "Deleting secret: $SECRET_NAME"
        aws secretsmanager delete-secret --secret-id "$SECRET_NAME" --force-
delete-without-recovery 2>/dev/null || true
    fi

    # Detach policies and delete runtime role if it exists
    if [ -n "${RUNTIME_ROLE_NAME:-}" ]; then
        echo "Deleting inline policies from runtime role: $RUNTIME_ROLE_NAME"
        for policy in $(aws iam list-role-policies --role-name
 "$RUNTIME_ROLE_NAME" --query 'PolicyNames[]' --output text 2>/dev/null || true);
 do
            aws iam delete-role-policy --role-name "$RUNTIME_ROLE_NAME" --policy-
name "$policy" 2>/dev/null || true
        done
        echo "Deleting IAM role: $RUNTIME_ROLE_NAME"
        aws iam delete-role --role-name "$RUNTIME_ROLE_NAME" 2>/dev/null || true
    fi

    # Detach policies and delete admin role if it exists
    if [ -n "${ADMIN_ROLE_NAME:-}" ]; then
        echo "Detaching policy from role: $ADMIN_ROLE_NAME"
        aws iam detach-role-policy --role-name "$ADMIN_ROLE_NAME" --policy-arn
 "arn:aws:iam::aws:policy/SecretsManagerReadWrite" 2>/dev/null || true

        for policy in $(aws iam list-role-policies --role-name "$ADMIN_ROLE_NAME"
 --query 'PolicyNames[]' --output text 2>/dev/null || true); do
            aws iam delete-role-policy --role-name "$ADMIN_ROLE_NAME" --policy-
name "$policy" 2>/dev/null || true
        done

        echo "Deleting IAM role: $ADMIN_ROLE_NAME"
        aws iam delete-role --role-name "$ADMIN_ROLE_NAME" 2>/dev/null || true
    fi

    echo "Cleanup completed."
```

```
}

# Trap to ensure cleanup on script exit
trap 'echo "Script interrupted. Running cleanup..."; cleanup_resources' INT TERM
 EXIT

# Generate random identifiers for resources
ADMIN_ROLE_NAME="SecretsManagerAdmin-$(generate_random_id)"
RUNTIME_ROLE_NAME="RoleToRetrieveSecretAtRuntime-$(generate_random_id)"
SECRET_NAME="MyAPIKey-$(generate_random_id)"

echo "Using the following resource names:"
echo "Admin Role: $ADMIN_ROLE_NAME"
echo "Runtime Role: $RUNTIME_ROLE_NAME"
echo "Secret Name: $SECRET_NAME"
echo ""

# Step 1: Create IAM roles
echo "Creating IAM roles..."

# Create assume role policy document
ASSUME_ROLE_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF
)

# Create the SecretsManagerAdmin role
echo "Creating admin role: $ADMIN_ROLE_NAME"
ADMIN_ROLE_OUTPUT=$(aws iam create-role \
    --role-name "$ADMIN_ROLE_NAME" \
    --assume-role-policy-document "$ASSUME_ROLE_POLICY" 2>&1)

check_error "$ADMIN_ROLE_OUTPUT" "create-role for admin"
```

```
echo "$ADMIN_ROLE_OUTPUT"
aws iam tag-role --role-name "$ADMIN_ROLE_NAME" --tags Key=project,Value=doc-
smith Key=tutorial,Value=aws-secrets-manager-gs

# Attach the SecretsManagerReadWrite policy to the admin role
echo "Attaching SecretsManagerReadWrite policy to admin role"
ATTACH_POLICY_OUTPUT=$(aws iam attach-role-policy \
    --role-name "$ADMIN_ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/SecretsManagerReadWrite" 2>&1)

check_error "$ATTACH_POLICY_OUTPUT" "attach-role-policy for admin"
echo "Policy attached successfully"

# Create the RoleToRetrieveSecretAtRuntime role
echo "Creating runtime role: $RUNTIME_ROLE_NAME"
RUNTIME_ROLE_OUTPUT=$(aws iam create-role \
    --role-name "$RUNTIME_ROLE_NAME" \
    --assume-role-policy-document "$ASSUME_ROLE_POLICY" 2>&1)

check_error "$RUNTIME_ROLE_OUTPUT" "create-role for runtime"
echo "$RUNTIME_ROLE_OUTPUT"
aws iam tag-role --role-name "$RUNTIME_ROLE_NAME" --tags Key=project,Value=doc-
smith Key=tutorial,Value=aws-secrets-manager-gs

# Wait for roles to be fully created
echo "Waiting for IAM roles to be fully created..."
sleep 10

# Step 2: Create a secret in AWS Secrets Manager
echo "Creating secret in AWS Secrets Manager..."

# Generate secure secret value using environment variable or secure method
# WARNING: In production, use secure methods to inject secrets (AWS CodeBuild,
 parameter store, etc.)
if [ -z "${TUTORIAL_SECRET_VALUE:-}" ]; then
    SECRET_VALUE=$(python3 -c "import json;
 print(json.dumps({'ClientID':'my_client_id','ClientSecret':__import__('secrets').token_u
else
    SECRET_VALUE="$TUTORIAL_SECRET_VALUE"
fi

CREATE_SECRET_OUTPUT=$(aws secretsmanager create-secret \
    --name "$SECRET_NAME" \
    --description "API key for my application" \
```

```
    --secret-string "$SECRET_VALUE" \
    --add-replica-regions 'Region=us-east-1' \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-secrets-manager-gs
 2>&1)

check_error "$CREATE_SECRET_OUTPUT" "create-secret"
echo "$CREATE_SECRET_OUTPUT"

# Get AWS account ID
echo "Getting AWS account ID..."
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text 2>&1)
check_error "$ACCOUNT_ID" "get-caller-identity"
echo "Account ID: $ACCOUNT_ID"

# Get secret ARN for precise resource policy
echo "Getting secret ARN..."
SECRET_ARN=$(aws secretsmanager describe-secret \
    --secret-id "$SECRET_NAME" \
    --query 'ARN' \
    --output text 2>&1)
check_error "$SECRET_ARN" "describe-secret"

# Add resource policy to the secret with least privilege
echo "Adding resource policy to secret..."
RESOURCE_POLICY=$(cat <<EOF
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRuntimeRoleReadSecret",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::$ACCOUNT_ID:role/$RUNTIME_ROLE_NAME"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "$SECRET_ARN",
            "Condition": {
                "StringEquals": {
                    "secretsmanager:VersionStage": "AWSCURRENT"
                }
            }
        }
    ]
}
```

```
EOF
)

PUT_POLICY_OUTPUT=$(aws secretsmanager put-resource-policy \
    --secret-id "$SECRET_NAME" \
    --resource-policy "$RESOURCE_POLICY" \
    --block-public-policy 2>&1)

check_error "$PUT_POLICY_OUTPUT" "put-resource-policy"
echo "Resource policy added successfully"

# Enable rotation policy recommendation
echo "Enabling secret metadata tags for rotation tracking..."
aws secretsmanager tag-resource \
    --secret-id "$SECRET_NAME" \
    --tags Key=Purpose,Value=Tutorial Key=AutoRotation,Value=Recommended 2>/dev/
null || true

# Step 3: Demonstrate retrieving the secret
echo "Retrieving the secret value (for demonstration purposes)..."
GET_SECRET_OUTPUT=$(aws secretsmanager get-secret-value \
    --secret-id "$SECRET_NAME" 2>&1)

check_error "$GET_SECRET_OUTPUT" "get-secret-value"
echo "Secret retrieved successfully. Secret metadata:"
echo "$GET_SECRET_OUTPUT" | jq '{ARN: .ARN, Name: .Name,
 LastUpdatedDate: .LastUpdatedDate, VersionIdsToStages: .VersionIdsToStages}' 2>/
dev/null || echo "Secret metadata retrieved (jq not available)"

# Step 4: Update the secret with new values
echo "Updating the secret with new values..."
UPDATE_SECRET_VALUE=$(python3 -c "import json;
 print(json.dumps({'ClientID':'my_new_client_id','ClientSecret':__import__('secrets').tok

UPDATE_SECRET_OUTPUT=$(aws secretsmanager update-secret \
    --secret-id "$SECRET_NAME" \
    --secret-string "$UPDATE_SECRET_VALUE" 2>&1)

check_error "$UPDATE_SECRET_OUTPUT" "update-secret"
echo "Secret updated successfully"

# Step 5: Verify the updated secret
echo "Verifying the updated secret..."
VERIFY_SECRET_OUTPUT=$(aws secretsmanager get-secret-value \
```

```
    --secret-id "$SECRET_NAME" 2>&1)

check_error "$VERIFY_SECRET_OUTPUT" "get-secret-value for verification"
echo "Updated secret retrieved successfully. Secret metadata:"
echo "$VERIFY_SECRET_OUTPUT" | jq '{ARN: .ARN, Name: .Name,
 LastUpdatedDate: .LastUpdatedDate, VersionIdsToStages: .VersionIdsToStages}' 2>/
dev/null || echo "Secret metadata retrieved (jq not available)"

# Step 6: Display rotation recommendations
echo ""
echo "Rotation Configuration Recommendations:"
echo "======================================="
DESCRIBE_OUTPUT=$(aws secretsmanager describe-secret --secret-id "$SECRET_NAME"
 2>&1)
if echo "$DESCRIBE_OUTPUT" | grep -q "RotationRules"; then
    echo "Current rotation configuration:"
    echo "$DESCRIBE_OUTPUT" | jq '.RotationRules' 2>/dev/null || echo "Rotation
 rules available"
else
    echo "No automatic rotation configured. Consider enabling rotation with:"
    echo "aws secretsmanager rotate-secret --secret-id $SECRET_NAME --rotation-
lambda-arn arn:aws:lambda:REGION:ACCOUNT:function:FUNCTION_NAME --rotation-rules
 AutomaticallyAfterDays=30"
fi

echo ""
echo "======================================================"
echo "Tutorial completed successfully!"
echo ""
echo "Summary of what we did:"
echo "1. Created IAM roles for managing and retrieving secrets"
echo "2. Created a secret in AWS Secrets Manager with secure generation"
echo "3. Added a least-privilege resource policy to control access to the secret"
echo "4. Retrieved the secret value (simulating application access)"
echo "5. Updated the secret with cryptographically secure values"
echo "6. Verified the updated secret"
echo ""
echo "Security best practices applied:"
echo "- Used cryptographically secure random ID generation"
echo "- Applied least-privilege resource policies with version stages"
echo "- Tagged resources for rotation tracking"
echo "- Blocked public access to secrets"
echo "- Used ARN-specific permissions instead of wildcards"
echo ""
```

```
echo "Next steps you might want to consider:"
echo "- Enable automatic secret rotation with AWS Lambda"
echo "- Implement secret caching in your application"
echo "- Set up CloudTrail logging for secret access auditing"
echo "- Use AWS CodeGuru Reviewer to find hardcoded secrets in your code"
echo "- For multi-region applications, replicate your secrets across regions"
echo "- Configure VPC endpoints for private access to Secrets Manager"
echo ""

echo "Script completed at $(date)"
exit 0
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateRole
  - CreateSecret
  - DeleteRole
  - DeleteSecret
  - DetachRolePolicy
  - GetCallerIdentity
  - GetSecretValue
  - PutResourcePolicy
  - UpdateSecret

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Run CPU stress tests on virtual machine instances using fault injection

The following code example shows how to:

- Create IAM roles
- Create a CloudWatch alarm
- Create an experiment template
- Run the experiment

- Verify the results
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS FIS CPU Stress Test Tutorial Script
# This script automates the steps in the AWS FIS CPU stress test tutorial

#    approach using epoch time calculations that work across all Linux
 distributions

# Set up logging
LOG_FILE="fis-tutorial-$(date +%Y%m%d-%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting AWS FIS CPU Stress Test Tutorial Script"
echo "Logging to $LOG_FILE"
echo "============================================="

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2

    if echo "$output" | grep -i "error" > /dev/null; then
        # Ignore specific expected errors
        if [[ "$cmd" == *"aws fis get-experiment"* ]] && [[ "$output" ==
 *"ConfigurationFailure"* ]]; then
            echo "Note: Experiment failed due to configuration issue. This is
 expected in some cases."
            return 0
```

```
        fi

        echo "ERROR: Command failed: $cmd"
        echo "Output: $output"
        cleanup_on_error
        exit 1
    fi
}

# Function to clean up resources on error
cleanup_on_error() {
    echo "Error encountered. Cleaning up resources..."

    if [ -n "$EXPERIMENT_ID" ]; then
        echo "Stopping experiment $EXPERIMENT_ID if running..."
        aws fis stop-experiment --id "$EXPERIMENT_ID" 2>/dev/null || true
    fi

    if [ -n "$TEMPLATE_ID" ]; then
        echo "Deleting experiment template $TEMPLATE_ID..."
        aws fis delete-experiment-template --id "$TEMPLATE_ID" || true
    fi

    if [ -n "$INSTANCE_ID" ]; then
        echo "Terminating EC2 instance $INSTANCE_ID..."
        aws ec2 terminate-instances --instance-ids "$INSTANCE_ID" || true
    fi

    if [ -n "$ALARM_NAME" ]; then
        echo "Deleting CloudWatch alarm $ALARM_NAME..."
        aws cloudwatch delete-alarms --alarm-names "$ALARM_NAME" || true
    fi

    if [ -n "$INSTANCE_PROFILE_NAME" ]; then
        echo "Removing role from instance profile..."
        aws iam remove-role-from-instance-profile --instance-profile-name
 "$INSTANCE_PROFILE_NAME" --role-name "$EC2_ROLE_NAME" || true

        echo "Deleting instance profile..."
        aws iam delete-instance-profile --instance-profile-name
 "$INSTANCE_PROFILE_NAME" || true
    fi

    if [ -n "$FIS_ROLE_NAME" ]; then
```

```
        echo "Deleting FIS role policy..."
        aws iam delete-role-policy --role-name "$FIS_ROLE_NAME" --policy-name
 "$FIS_POLICY_NAME" || true

        echo "Deleting FIS role..."
        aws iam delete-role --role-name "$FIS_ROLE_NAME" || true
    fi

    if [ -n "$EC2_ROLE_NAME" ]; then
        echo "Detaching policy from EC2 role..."
        aws iam detach-role-policy --role-name "$EC2_ROLE_NAME" --policy-arn
 "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore" || true

        echo "Deleting EC2 role..."
        aws iam delete-role --role-name "$EC2_ROLE_NAME" || true
    fi

    echo "Cleanup completed."
}

# Generate unique identifiers for resources
TIMESTAMP=$(date +%Y%m%d%H%M%S)
FIS_ROLE_NAME="FISRole-${TIMESTAMP}"
FIS_POLICY_NAME="FISPolicy-${TIMESTAMP}"
EC2_ROLE_NAME="EC2SSMRole-${TIMESTAMP}"
INSTANCE_PROFILE_NAME="EC2SSMProfile-${TIMESTAMP}"
ALARM_NAME="FIS-CPU-Alarm-${TIMESTAMP}"

# Track created resources
CREATED_RESOURCES=()

echo "Step 1: Creating IAM role for AWS FIS"
# Create trust policy file for AWS FIS
cat > fis-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "fis.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
```

```
  ]
}
EOF

# Create IAM role for FIS
echo "Creating IAM role $FIS_ROLE_NAME for AWS FIS..."
FIS_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$FIS_ROLE_NAME" \
  --assume-role-policy-document file://fis-trust-policy.json)
check_error "$FIS_ROLE_OUTPUT" "aws iam create-role"
aws iam tag-role --role-name "$FIS_ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-fault-injection-service-gs
CREATED_RESOURCES+=("IAM Role: $FIS_ROLE_NAME")

# Create policy document for SSM actions
cat > fis-ssm-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:SendCommand",
        "ssm:ListCommands",
        "ssm:ListCommandInvocations"
      ],
      "Resource": "*"
    }
  ]
}
EOF

# Attach policy to the role
echo "Attaching policy $FIS_POLICY_NAME to role $FIS_ROLE_NAME..."
FIS_POLICY_OUTPUT=$(aws iam put-role-policy \
  --role-name "$FIS_ROLE_NAME" \
  --policy-name "$FIS_POLICY_NAME" \
  --policy-document file://fis-ssm-policy.json)
check_error "$FIS_POLICY_OUTPUT" "aws iam put-role-policy"
CREATED_RESOURCES+=("IAM Policy: $FIS_POLICY_NAME attached to $FIS_ROLE_NAME")

echo "Step 2: Creating IAM role for EC2 instance with SSM permissions"
# Create trust policy file for EC2
cat > ec2-trust-policy.json << 'EOF'
```

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role for EC2
echo "Creating IAM role $EC2_ROLE_NAME for EC2 instance..."
EC2_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$EC2_ROLE_NAME" \
  --assume-role-policy-document file://ec2-trust-policy.json)
check_error "$EC2_ROLE_OUTPUT" "aws iam create-role"
aws iam tag-role --role-name "$EC2_ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-fault-injection-service-gs
CREATED_RESOURCES+=("IAM Role: $EC2_ROLE_NAME")

# Attach SSM policy to the EC2 role
echo "Attaching AmazonSSMManagedInstanceCore policy to role $EC2_ROLE_NAME..."
EC2_POLICY_OUTPUT=$(aws iam attach-role-policy \
  --role-name "$EC2_ROLE_NAME" \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore)
check_error "$EC2_POLICY_OUTPUT" "aws iam attach-role-policy"
CREATED_RESOURCES+=("IAM Policy: AmazonSSMManagedInstanceCore attached to
 $EC2_ROLE_NAME")

# Create instance profile
echo "Creating instance profile $INSTANCE_PROFILE_NAME..."
PROFILE_OUTPUT=$(aws iam create-instance-profile \
  --instance-profile-name "$INSTANCE_PROFILE_NAME")
check_error "$PROFILE_OUTPUT" "aws iam create-instance-profile"
CREATED_RESOURCES+=("IAM Instance Profile: $INSTANCE_PROFILE_NAME")

# Add role to instance profile
echo "Adding role $EC2_ROLE_NAME to instance profile $INSTANCE_PROFILE_NAME..."
ADD_ROLE_OUTPUT=$(aws iam add-role-to-instance-profile \
  --instance-profile-name "$INSTANCE_PROFILE_NAME" \
```

```
  --role-name "$EC2_ROLE_NAME")
check_error "$ADD_ROLE_OUTPUT" "aws iam add-role-to-instance-profile"

# Wait for role to propagate
echo "Waiting for IAM role to propagate..."
sleep 10

echo "Step 3: Launching EC2 instance"
# Get the latest Amazon Linux 2 AMI ID
echo "Finding latest Amazon Linux 2 AMI..."
AMI_ID=$(aws ec2 describe-images \
  --owners amazon \
  --filters "Name=name,Values=amzn2-ami-hvm-*-x86_64-gp2"
 "Name=state,Values=available" \
  --query "sort_by(Images, &CreationDate)[-1].ImageId" \
  --output text)
check_error "$AMI_ID" "aws ec2 describe-images"
echo "Using AMI: $AMI_ID"

# Launch EC2 instance
echo "Launching EC2 instance with AMI $AMI_ID..."
INSTANCE_OUTPUT=$(aws ec2 run-instances \
  --image-id "$AMI_ID" \
  --instance-type t2.micro \
  --iam-instance-profile Name="$INSTANCE_PROFILE_NAME" \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=FIS-Test-
Instance},{Key=project,Value=doc-smith},{Key=tutorial,Value=aws-fault-injection-
service-gs}]')
check_error "$INSTANCE_OUTPUT" "aws ec2 run-instances"

# Get instance ID
INSTANCE_ID=$(echo "$INSTANCE_OUTPUT" | grep -i "InstanceId" | head -1 | awk -
F'"' '{print $4}')
if [ -z "$INSTANCE_ID" ]; then
    echo "Failed to get instance ID"
    cleanup_on_error
    exit 1
fi
echo "Launched instance: $INSTANCE_ID"
CREATED_RESOURCES+=("EC2 Instance: $INSTANCE_ID")

# Enable detailed monitoring
echo "Enabling detailed monitoring for instance $INSTANCE_ID..."
MONITOR_OUTPUT=$(aws ec2 monitor-instances --instance-ids "$INSTANCE_ID")
```

```
check_error "$MONITOR_OUTPUT" "aws ec2 monitor-instances"

# Wait for instance to be running and status checks to pass
echo "Waiting for instance to be ready..."
aws ec2 wait instance-running --instance-ids "$INSTANCE_ID"
aws ec2 wait instance-status-ok --instance-ids "$INSTANCE_ID"
echo "Instance is ready"

echo "Step 4: Creating CloudWatch alarm for CPU utilization"
# Create CloudWatch alarm
echo "Creating CloudWatch alarm $ALARM_NAME..."
ALARM_OUTPUT=$(aws cloudwatch put-metric-alarm \
  --alarm-name "$ALARM_NAME" \
  --alarm-description "Alarm when CPU exceeds 50%" \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Maximum \
  --period 60 \
  --threshold 50 \
  --comparison-operator GreaterThanOrEqualToThreshold \
  --dimensions "Name=InstanceId,Value=$INSTANCE_ID" \
  --evaluation-periods 1)
check_error "$ALARM_OUTPUT" "aws cloudwatch put-metric-alarm"
CREATED_RESOURCES+=("CloudWatch Alarm: $ALARM_NAME")

# Get the alarm ARN
echo "Getting CloudWatch alarm ARN..."
ALARM_ARN_OUTPUT=$(aws cloudwatch describe-alarms \
  --alarm-names "$ALARM_NAME")
check_error "$ALARM_ARN_OUTPUT" "aws cloudwatch describe-alarms"
ALARM_ARN=$(echo "$ALARM_ARN_OUTPUT" | grep -i "AlarmArn" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$ALARM_ARN" ]; then
    echo "Failed to get alarm ARN"
    cleanup_on_error
    exit 1
fi
echo "Alarm ARN: $ALARM_ARN"

# Wait for the alarm to initialize and reach OK state
echo "Waiting for CloudWatch alarm to initialize (60 seconds)..."
sleep 60

# Check alarm state
```

```
echo "Checking alarm state..."
ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms \
  --alarm-names "$ALARM_NAME")
ALARM_STATE=$(echo "$ALARM_STATE_OUTPUT" | grep -i "StateValue" | head -1 | awk -
F'"' '{print $4}')
echo "Current alarm state: $ALARM_STATE"

# If alarm is not in OK state, wait longer or generate some baseline metrics
if [ "$ALARM_STATE" != "OK" ]; then
    echo "Alarm not in OK state. Waiting for alarm to stabilize (additional 60
 seconds)..."
    sleep 60

    # Check alarm state again
    ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms \
      --alarm-names "$ALARM_NAME")
    ALARM_STATE=$(echo "$ALARM_STATE_OUTPUT" | grep -i "StateValue" | head -1 |
 awk -F'"' '{print $4}')
    echo "Updated alarm state: $ALARM_STATE"

    if [ "$ALARM_STATE" != "OK" ]; then
        echo "Warning: Alarm still not in OK state. Experiment may fail to
 start."
    fi
fi

echo "Step 5: Creating AWS FIS experiment template"
# Get the IAM role ARN
echo "Getting IAM role ARN for $FIS_ROLE_NAME..."
ROLE_ARN_OUTPUT=$(aws iam get-role \
  --role-name "$FIS_ROLE_NAME")
check_error "$ROLE_ARN_OUTPUT" "aws iam get-role"
ROLE_ARN=$(echo "$ROLE_ARN_OUTPUT" | grep -i "Arn" | head -1 | awk -F'"' '{print
 $4}')
if [ -z "$ROLE_ARN" ]; then
    echo "Failed to get role ARN"
    cleanup_on_error
    exit 1
fi
echo "Role ARN: $ROLE_ARN"

# Get account ID and region
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
REGION=$(aws configure get region)
```

```
if [ -z "$REGION" ]; then
    REGION="us-east-1"  # Default to us-east-1 if region not set
fi
INSTANCE_ARN="arn:aws:ec2:${REGION}:${ACCOUNT_ID}:instance/${INSTANCE_ID}"
echo "Instance ARN: $INSTANCE_ARN"

# Create experiment template - Fixed JSON escaping issue
cat > experiment-template.json << EOF
{
  "description": "Test CPU stress predefined SSM document",
  "targets": {
    "testInstance": {
      "resourceType": "aws:ec2:instance",
      "resourceArns": ["$INSTANCE_ARN"],
      "selectionMode": "ALL"
    }
  },
  "actions": {
    "runCpuStress": {
      "actionId": "aws:ssm:send-command",
      "parameters": {
        "documentArn": "arn:aws:ssm:$REGION::document/AWSFIS-Run-CPU-Stress",
        "documentParameters": "{\"DurationSeconds\":\"120\"}",
        "duration": "PT5M"
      },
      "targets": {
        "Instances": "testInstance"
      }
    }
  },
  "stopConditions": [
    {
      "source": "aws:cloudwatch:alarm",
      "value": "$ALARM_ARN"
    }
  ],
  "roleArn": "$ROLE_ARN",
  "tags": {
    "Name": "FIS-CPU-Stress-Experiment",
    "project": "doc-smith",
    "tutorial": "aws-fault-injection-service-gs"
  }
}
EOF
```

```
# Create experiment template
echo "Creating AWS FIS experiment template..."
TEMPLATE_OUTPUT=$(aws fis create-experiment-template --cli-input-json file://
experiment-template.json)
check_error "$TEMPLATE_OUTPUT" "aws fis create-experiment-template"
TEMPLATE_ID=$(echo "$TEMPLATE_OUTPUT" | grep -i "id" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$TEMPLATE_ID" ]; then
    echo "Failed to get template ID"
    cleanup_on_error
    exit 1
fi
echo "Experiment template created with ID: $TEMPLATE_ID"
CREATED_RESOURCES+=("FIS Experiment Template: $TEMPLATE_ID")

echo "Step 6: Starting the experiment"
# Start the experiment
echo "Starting AWS FIS experiment using template $TEMPLATE_ID..."
EXPERIMENT_OUTPUT=$(aws fis start-experiment \
  --experiment-template-id "$TEMPLATE_ID" \
  --tags '{"Name": "FIS-CPU-Stress-Run"}')
check_error "$EXPERIMENT_OUTPUT" "aws fis start-experiment"
EXPERIMENT_ID=$(echo "$EXPERIMENT_OUTPUT" | grep -i "id" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$EXPERIMENT_ID" ]; then
    echo "Failed to get experiment ID"
    cleanup_on_error
    exit 1
fi
echo "Experiment started with ID: $EXPERIMENT_ID"
CREATED_RESOURCES+=("FIS Experiment: $EXPERIMENT_ID")

echo "Step 7: Tracking experiment progress"
# Track experiment progress
echo "Tracking experiment progress..."
MAX_CHECKS=30
CHECK_COUNT=0
EXPERIMENT_STATE=""

while [ $CHECK_COUNT -lt $MAX_CHECKS ]; do
    EXPERIMENT_INFO=$(aws fis get-experiment --id "$EXPERIMENT_ID")
    # Don't check for errors here, as we expect some experiments to fail
```

```
    EXPERIMENT_STATE=$(echo "$EXPERIMENT_INFO" | grep -i "status" | head -1 | awk
 -F'"' '{print $4}')
    echo "Experiment state: $EXPERIMENT_STATE"

    if [ "$EXPERIMENT_STATE" == "completed" ] || [ "$EXPERIMENT_STATE" ==
 "stopped" ] || [ "$EXPERIMENT_STATE" == "failed" ]; then
        # Show the reason for the state
        REASON=$(echo "$EXPERIMENT_INFO" | grep -i "reason" | head -1 | awk -F'"'
 '{print $4}')
        if [ -n "$REASON" ]; then
            echo "Reason: $REASON"
        fi
        break
    fi

    echo "Waiting 10 seconds before checking again..."
    sleep 10
    CHECK_COUNT=$((CHECK_COUNT + 1))
done

if [ $CHECK_COUNT -eq $MAX_CHECKS ]; then
    echo "Experiment is taking longer than expected. You can check its status
 later using:"
    echo "aws fis get-experiment --id $EXPERIMENT_ID"
fi

echo "Step 8: Verifying experiment results"
# Check CloudWatch alarm state
echo "Checking CloudWatch alarm state..."
ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms --alarm-names "$ALARM_NAME")
check_error "$ALARM_STATE_OUTPUT" "aws cloudwatch describe-alarms"
echo "$ALARM_STATE_OUTPUT"

# Get CPU utilization metrics
echo "Getting CPU utilization metrics..."
END_TIME=$(date -u +"%Y-%m-%dT%H:%M:%SZ")

# FIXED: Cross-platform compatible way to calculate time 10 minutes ago
# This approach uses epoch seconds and basic arithmetic which works on all Linux
 distributions
CURRENT_EPOCH=$(date +%s)
TEN_MINUTES_AGO_EPOCH=$((CURRENT_EPOCH - 600))
START_TIME=$(date -u -d "@$TEN_MINUTES_AGO_EPOCH" +"%Y-%m-%dT%H:%M:%SZ" 2>/dev/
null || date -u -r "$TEN_MINUTES_AGO_EPOCH" +"%Y-%m-%dT%H:%M:%SZ")
```

```
# Create metric query file
cat > metric-query.json << EOF
[
  {
    "Id": "cpu",
    "MetricStat": {
      "Metric": {
        "Namespace": "AWS/EC2",
        "MetricName": "CPUUtilization",
        "Dimensions": [
          {
            "Name": "InstanceId",
            "Value": "$INSTANCE_ID"
          }
        ]
      },
      "Period": 60,
      "Stat": "Maximum"
    }
  }
]
EOF

METRICS_OUTPUT=$(aws cloudwatch get-metric-data \
  --start-time "$START_TIME" \
  --end-time "$END_TIME" \
  --metric-data-queries file://metric-query.json)
check_error "$METRICS_OUTPUT" "aws cloudwatch get-metric-data"
echo "CPU Utilization Metrics:"
echo "$METRICS_OUTPUT"

# Display summary of created resources
echo ""
echo "==========================================="
echo "RESOURCES CREATED"
echo "==========================================="
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "- $resource"
done
echo "==========================================="

# Prompt for cleanup
echo ""
```

```
echo "============================================"
echo "CLEANUP CONFIRMATION"
echo "============================================"
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    echo "Starting cleanup process..."

    # Stop experiment if still running
    if [ "$EXPERIMENT_STATE" != "completed" ] && [ "$EXPERIMENT_STATE" !=
 "stopped" ] && [ "$EXPERIMENT_STATE" != "failed" ]; then
        echo "Stopping experiment $EXPERIMENT_ID..."
        STOP_OUTPUT=$(aws fis stop-experiment --id "$EXPERIMENT_ID")
        check_error "$STOP_OUTPUT" "aws fis stop-experiment"
        echo "Waiting for experiment to stop..."
        sleep 10
    fi

    # Delete experiment template
    echo "Deleting experiment template $TEMPLATE_ID..."
    DELETE_TEMPLATE_OUTPUT=$(aws fis delete-experiment-template --id
 "$TEMPLATE_ID")
    check_error "$DELETE_TEMPLATE_OUTPUT" "aws fis delete-experiment-template"

    # Delete CloudWatch alarm
    echo "Deleting CloudWatch alarm $ALARM_NAME..."
    DELETE_ALARM_OUTPUT=$(aws cloudwatch delete-alarms --alarm-names
 "$ALARM_NAME")
    check_error "$DELETE_ALARM_OUTPUT" "aws cloudwatch delete-alarms"

    # Terminate EC2 instance
    echo "Terminating EC2 instance $INSTANCE_ID..."
    TERMINATE_OUTPUT=$(aws ec2 terminate-instances --instance-ids "$INSTANCE_ID")
    check_error "$TERMINATE_OUTPUT" "aws ec2 terminate-instances"
    echo "Waiting for instance to terminate..."
    aws ec2 wait instance-terminated --instance-ids "$INSTANCE_ID"

    # Clean up IAM resources
    echo "Removing role from instance profile..."
    REMOVE_ROLE_OUTPUT=$(aws iam remove-role-from-instance-profile \
      --instance-profile-name "$INSTANCE_PROFILE_NAME" \
      --role-name "$EC2_ROLE_NAME")
    check_error "$REMOVE_ROLE_OUTPUT" "aws iam remove-role-from-instance-profile"
```

```
    echo "Deleting instance profile..."
    DELETE_PROFILE_OUTPUT=$(aws iam delete-instance-profile \
      --instance-profile-name "$INSTANCE_PROFILE_NAME")
    check_error "$DELETE_PROFILE_OUTPUT" "aws iam delete-instance-profile"

    echo "Deleting FIS role policy..."
    DELETE_POLICY_OUTPUT=$(aws iam delete-role-policy \
      --role-name "$FIS_ROLE_NAME" \
      --policy-name "$FIS_POLICY_NAME")
    check_error "$DELETE_POLICY_OUTPUT" "aws iam delete-role-policy"

    echo "Detaching policy from EC2 role..."
    DETACH_POLICY_OUTPUT=$(aws iam detach-role-policy \
      --role-name "$EC2_ROLE_NAME" \
      --policy-arn "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore")
    check_error "$DETACH_POLICY_OUTPUT" "aws iam detach-role-policy"

    echo "Deleting FIS role..."
    DELETE_FIS_ROLE_OUTPUT=$(aws iam delete-role \
      --role-name "$FIS_ROLE_NAME")
    check_error "$DELETE_FIS_ROLE_OUTPUT" "aws iam delete-role"

    echo "Deleting EC2 role..."
    DELETE_EC2_ROLE_OUTPUT=$(aws iam delete-role \
      --role-name "$EC2_ROLE_NAME")
    check_error "$DELETE_EC2_ROLE_OUTPUT" "aws iam delete-role"

    # Clean up temporary files
    echo "Cleaning up temporary files..."
    rm -f fis-trust-policy.json ec2-trust-policy.json fis-ssm-policy.json
 experiment-template.json metric-query.json

    echo "Cleanup completed successfully."
else
    echo "Cleanup skipped. Resources will remain in your AWS account."
    echo "You can manually clean up the resources listed above."
fi

echo ""
echo "Script execution completed."
echo "Log file: $LOG_FILE"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddRoleToInstanceProfile
  - AttachRolePolicy
  - CreateExperimentTemplate
  - CreateInstanceProfile
  - CreateRole
  - DeleteAlarms
  - DeleteExperimentTemplate
  - DeleteInstanceProfile
  - DeleteRole
  - DeleteRolePolicy
  - DescribeAlarms
  - DescribeImages
  - DetachRolePolicy
  - GetCallerIdentity
  - GetExperiment
  - GetMetricData
  - GetRole
  - MonitorInstances
  - PutMetricAlarm
  - PutRolePolicy
  - RemoveRoleFromInstanceProfile
  - RunInstances
  - StartExperiment
  - StopExperiment
  - TerminateInstances
  - Wait

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Setting up systems management

The following code example shows how to:

- Create IAM permissions for Systems Manager
- Create an IAM role for Systems Manager
- Configure Systems Manager
- Verify the setup
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS Systems Manager Setup Script
# This script sets up AWS Systems Manager for a single account and region
#
# Version 17 fixes:
# 1. Added cloudformation.amazonaws.com to the IAM role trust policy
# 2. Systems Manager Quick Setup uses CloudFormation for deployments, so the role
 must trust CloudFormation service

# Initialize log file
LOG_FILE="ssm_setup_$(date +%Y%m%d_%H%M%S).log"
UNIQUE_ID=$(openssl rand -hex 4)
echo "Starting AWS Systems Manager setup at $(date)" > "$LOG_FILE"

# Function to log commands and their outputs with immediate terminal display
log_cmd() {
    echo "$(date): Running command: $1" | tee -a "$LOG_FILE"
    local output
    output=$(eval "$1" 2>&1)
```

```
    local status=$?
    echo "$output" | tee -a "$LOG_FILE"
    return $status
}

# Function to check for errors in command output
check_error() {
    local cmd_output="$1"
    local cmd_status="$2"
    local error_msg="$3"

    if [[ $cmd_status -ne 0 || "$cmd_output" =~ [Ee][Rr][Rr][Oo][Rr] ]]; then
        echo "ERROR: $error_msg" | tee -a "$LOG_FILE"
        echo "Command output: $cmd_output" | tee -a "$LOG_FILE"
        cleanup_on_error
        exit 1
    fi
}

# Array to track created resources for cleanup
declare -a CREATED_RESOURCES

# Function to add a resource to the tracking array
track_resource() {
    local resource_type="$1"
    local resource_id="$2"
    CREATED_RESOURCES+=("$resource_type:$resource_id")
    echo "Tracked resource: $resource_type:$resource_id" | tee -a "$LOG_FILE"
}

# Function to clean up resources on error
cleanup_on_error() {
    echo "" | tee -a "$LOG_FILE"
    echo "========================================" | tee -a "$LOG_FILE"
    echo "ERROR OCCURRED - CLEANING UP RESOURCES" | tee -a "$LOG_FILE"
    echo "========================================" | tee -a "$LOG_FILE"
    echo "The following resources were created:" | tee -a "$LOG_FILE"

    # Display resources in reverse order
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        echo "${CREATED_RESOURCES[$i]}" | tee -a "$LOG_FILE"
    done

    echo "" | tee -a "$LOG_FILE"
```

```
    echo "Attempting to clean up resources..." | tee -a "$LOG_FILE"

    # Clean up resources in reverse order
    cleanup_resources
}

# Function to clean up all created resources
cleanup_resources() {
    # Process resources in reverse order (last created, first deleted)
    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        IFS=':' read -r resource_type resource_id <<< "${CREATED_RESOURCES[$i]}"

        echo "Deleting $resource_type: $resource_id" | tee -a "$LOG_FILE"

        case "$resource_type" in
            "IAM_POLICY")
                # Delete the policy (detachment should have been handled when the
 role was deleted)
                log_cmd "aws iam delete-policy --policy-arn $resource_id" || true
                ;;
            "IAM_ROLE")
                # Detach all policies from the role first
                if [[ -n "$POLICY_ARN" ]]; then
                    log_cmd "aws iam detach-role-policy --role-name $resource_id
 --policy-arn $POLICY_ARN" || true
                fi

                # Delete the role
                log_cmd "aws iam delete-role --role-name $resource_id" || true
                ;;
            "SSM_CONFIG_MANAGER")
                log_cmd "aws ssm-quicksetup delete-configuration-manager --
manager-arn $resource_id" || true
                ;;
            *)
                echo "Unknown resource type: $resource_type, cannot delete
 automatically" | tee -a "$LOG_FILE"
                ;;
        esac
    done

    echo "Cleanup completed" | tee -a "$LOG_FILE"

    # Clean up temporary files
```

```
    rm -f ssm-onboarding-policy.json trust-policy.json ssm-config.json 2>/dev/
null || true
}

# Main script execution
echo "AWS Systems Manager Setup Script"
echo "================================"
echo "This script will set up AWS Systems Manager for a single account and
 region."
echo "It will create IAM policies and roles, then enable Systems Manager
 features."
echo ""

# Get the current AWS region
CURRENT_REGION=$(aws configure get region)
if [[ -z "$CURRENT_REGION" ]]; then
    echo "No AWS region configured. Please specify a region:"
    CURRENT_REGION="${AWS_DEFAULT_REGION:-us-west-2}"
    if [[ -z "$CURRENT_REGION" ]]; then
        echo "ERROR: A region must be specified" | tee -a "$LOG_FILE"
        exit 1
    fi
fi

echo "Using AWS region: $CURRENT_REGION" | tee -a "$LOG_FILE"

# Step 1: Create IAM policy for Systems Manager onboarding
echo "Step 1: Creating IAM policy for Systems Manager onboarding..."

# Create policy document
cat > ssm-onboarding-policy.json << 'EOF'
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Sid": "QuickSetupActions",
       "Effect": "Allow",
       "Action": [
         "ssm-quicksetup:*"
       ],
       "Resource": "*"
     },
     {
       "Sid": "SsmReadOnly",
```

```
      "Effect": "Allow",
      "Action": [
        "ssm:DescribeAutomationExecutions",
        "ssm:GetAutomationExecution",
        "ssm:ListAssociations",
        "ssm:DescribeAssociation",
        "ssm:ListDocuments",
        "ssm:ListResourceDataSync",
        "ssm:DescribePatchBaselines",
        "ssm:GetPatchBaseline",
        "ssm:DescribeMaintenanceWindows",
        "ssm:DescribeMaintenanceWindowTasks"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SsmDocument",
      "Effect": "Allow",
      "Action": [
        "ssm:GetDocument",
        "ssm:DescribeDocument"
      ],
      "Resource": [
        "arn:aws:ssm:*:*:document/AWSQuickSetupType-*",
        "arn:aws:ssm:*:*:document/AWS-EnableExplorer"
      ]
    },
    {
      "Sid": "SsmEnableExplorer",
      "Effect": "Allow",
      "Action": "ssm:StartAutomationExecution",
      "Resource": "arn:aws:ssm:*:*:automation-definition/AWS-EnableExplorer:*"
    },
    {
      "Sid": "SsmExplorerRds",
      "Effect": "Allow",
      "Action": [
        "ssm:GetOpsSummary",
        "ssm:CreateResourceDataSync",
        "ssm:UpdateResourceDataSync"
      ],
      "Resource": "arn:aws:ssm:*:*:resource-data-sync/AWS-QuickSetup-*"
    },
    {
```

```
        "Sid": "OrgsReadOnly",
        "Effect": "Allow",
        "Action": [
          "organizations:DescribeAccount",
          "organizations:DescribeOrganization",
          "organizations:ListDelegatedAdministrators",
          "organizations:ListRoots",
          "organizations:ListParents",
          "organizations:ListOrganizationalUnitsForParent",
          "organizations:DescribeOrganizationalUnit",
          "organizations:ListAWSServiceAccessForOrganization"
        ],
        "Resource": "*"
      },
      {
        "Sid": "OrgsAdministration",
        "Effect": "Allow",
        "Action": [
          "organizations:EnableAWSServiceAccess",
          "organizations:RegisterDelegatedAdministrator",
          "organizations:DeregisterDelegatedAdministrator"
        ],
        "Resource": "*",
        "Condition": {
          "StringEquals": {
            "organizations:ServicePrincipal": [
              "ssm.amazonaws.com",
              "ssm-quicksetup.amazonaws.com",
              "member.org.stacksets.cloudformation.amazonaws.com",
              "resource-explorer-2.amazonaws.com"
            ]
          }
        }
      },
      {
        "Sid": "CfnReadOnly",
        "Effect": "Allow",
        "Action": [
          "cloudformation:ListStacks",
          "cloudformation:DescribeStacks",
          "cloudformation:ListStackSets",
          "cloudformation:DescribeOrganizationsAccess"
        ],
        "Resource": "*"
```

```
    },
    {
      "Sid": "OrgCfnAccess",
      "Effect": "Allow",
      "Action": [
        "cloudformation:ActivateOrganizationsAccess"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CfnStackActions",
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeStackResources",
        "cloudformation:DescribeStackEvents",
        "cloudformation:GetTemplate",
        "cloudformation:RollbackStack",
        "cloudformation:TagResource",
        "cloudformation:UntagResource",
        "cloudformation:UpdateStack"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/StackSet-AWS-QuickSetup-*",
        "arn:aws:cloudformation:*:*:stack/AWS-QuickSetup-*",
        "arn:aws:cloudformation:*:*:type/resource/*"
      ]
    },
    {
      "Sid": "CfnStackSetActions",
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateStackInstances",
        "cloudformation:CreateStackSet",
        "cloudformation:DeleteStackInstances",
        "cloudformation:DeleteStackSet",
        "cloudformation:DescribeStackInstance",
        "cloudformation:DetectStackSetDrift",
        "cloudformation:ListStackInstanceResourceDrifts",
        "cloudformation:DescribeStackSet",
        "cloudformation:DescribeStackSetOperation",
        "cloudformation:ListStackInstances",
        "cloudformation:ListStackSetOperations",
```

```
        "cloudformation:ListStackSetOperationResults",
        "cloudformation:TagResource",
        "cloudformation:UntagResource",
        "cloudformation:UpdateStackSet"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stackset/AWS-QuickSetup-*",
        "arn:aws:cloudformation:*:*:type/resource/*",
        "arn:aws:cloudformation:*:*:stackset-target/AWS-QuickSetup-*:*"
      ]
    },
    {
      "Sid": "ValidationReadonlyActions",
      "Effect": "Allow",
      "Action": [
        "iam:ListRoles",
        "iam:GetRole"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IamRolesMgmt",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:GetRole",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:GetRolePolicy",
        "iam:ListRolePolicies"
      ],
      "Resource": [
        "arn:aws:iam::*:role/AWS-QuickSetup-*",
        "arn:aws:iam::*:role/service-role/AWS-QuickSetup-*"
      ]
    },
    {
      "Sid": "IamPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
```

```
          "arn:aws:iam::*:role/AWS-QuickSetup-*",
          "arn:aws:iam::*:role/service-role/AWS-QuickSetup-*"
        ],
        "Condition": {
          "StringEquals": {
            "iam:PassedToService": [
              "ssm.amazonaws.com",
              "ssm-quicksetup.amazonaws.com",
              "cloudformation.amazonaws.com"
            ]
          }
        }
      },
      {
        "Sid": "IamRolesPoliciesMgmt",
        "Effect": "Allow",
        "Action": [
          "iam:AttachRolePolicy",
          "iam:DetachRolePolicy"
        ],
        "Resource": [
          "arn:aws:iam::*:role/AWS-QuickSetup-*",
          "arn:aws:iam::*:role/service-role/AWS-QuickSetup-*"
        ],
        "Condition": {
          "ArnEquals": {
            "iam:PolicyARN": [
              "arn:aws:iam::aws:policy/
AWSSystemsManagerEnableExplorerExecutionPolicy",
              "arn:aws:iam::aws:policy/AWSQuickSetupSSMDeploymentRolePolicy"
            ]
          }
        }
      },
      {
        "Sid": "CfnStackSetsSLR",
        "Effect": "Allow",
        "Action": [
          "iam:CreateServiceLinkedRole"
        ],
        "Resource": [
          "arn:aws:iam::*:role/aws-service-role/
stacksets.cloudformation.amazonaws.com/
AWSServiceRoleForCloudFormationStackSetsOrgAdmin",
```

```
        "arn:aws:iam::*:role/aws-service-role/ssm.amazonaws.com/
AWSServiceRoleForAmazonSSM",
        "arn:aws:iam::*:role/aws-service-role/
accountdiscovery.ssm.amazonaws.com/AWSServiceRoleForAmazonSSM_AccountDiscovery",
        "arn:aws:iam::*:role/aws-service-role/ssm-quicksetup.amazonaws.com/
AWSServiceRoleForSSMQuickSetup",
        "arn:aws:iam::*:role/aws-service-role/resource-explorer-2.amazonaws.com/
AWSServiceRoleForResourceExplorer"
      ]
    }
  ]
}
EOF

# Create the IAM policy
POLICY_OUTPUT=$(log_cmd "aws iam create-policy --policy-name SSMOnboardingPolicy-
$UNIQUE_ID --policy-document file://ssm-onboarding-policy.json --output json")
POLICY_STATUS=$?
check_error "$POLICY_OUTPUT" $POLICY_STATUS "Failed to create IAM policy"

# Extract the policy ARN
POLICY_ARN=$(echo "$POLICY_OUTPUT" | grep -o 'arn:aws:iam::[0-9]*:policy/
SSMOnboardingPolicy-[a-f0-9]*')
if [[ -z "$POLICY_ARN" ]]; then
    echo "ERROR: Failed to extract policy ARN" | tee -a "$LOG_FILE"
    exit 1
fi

# Track the created policy
track_resource "IAM_POLICY" "$POLICY_ARN"
aws iam tag-policy --policy-arn "$POLICY_ARN" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-systems-manager-gs

echo "Created policy: $POLICY_ARN" | tee -a "$LOG_FILE"

# Step 2: Create and configure IAM role for Systems Manager
echo ""
echo "Step 2: Creating IAM role for Systems Manager..."

# Get current user name
USER_OUTPUT=$(log_cmd "aws sts get-caller-identity --output json")
USER_STATUS=$?
check_error "$USER_OUTPUT" $USER_STATUS "Failed to get caller identity"
```

```
# Extract account ID
ACCOUNT_ID=$(echo "$USER_OUTPUT" | grep -o '"Account": "[0-9]*"' | cut -d'"' -f4)
if [[ -z "$ACCOUNT_ID" ]]; then
    echo "ERROR: Failed to extract account ID" | tee -a "$LOG_FILE"
    exit 1
fi

# Generate a unique role name
ROLE_NAME="SSMTutorialRole-$(openssl rand -hex 4)"

# Create trust policy for the role - FIXED: Added cloudformation.amazonaws.com
cat > trust-policy.json << 'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com",
                    "ssm-quicksetup.amazonaws.com",
                    "cloudformation.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::ACCOUNT_ID:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF

# Replace ACCOUNT_ID placeholder in trust policy
sed -i "s/ACCOUNT_ID/$ACCOUNT_ID/g" trust-policy.json

# Create the IAM role
ROLE_OUTPUT=$(log_cmd "aws iam create-role --role-name $ROLE_NAME --assume-role-
policy-document file://trust-policy.json --description 'Role for Systems Manager
 tutorial' --output json")
```

```
ROLE_STATUS=$?
check_error "$ROLE_OUTPUT" $ROLE_STATUS "Failed to create IAM role"

# Extract the role ARN
ROLE_ARN=$(echo "$ROLE_OUTPUT" | grep -o 'arn:aws:iam::[0-9]*:role/[^"]*')
if [[ -z "$ROLE_ARN" ]]; then
    echo "ERROR: Failed to extract role ARN" | tee -a "$LOG_FILE"
    cleanup_on_error
    exit 1
fi

# Track the created role
track_resource "IAM_ROLE" "$ROLE_NAME"
aws iam tag-role --role-name "$ROLE_NAME" \
  --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-systems-manager-gs

echo "Created IAM role: $ROLE_NAME" | tee -a "$LOG_FILE"
echo "Role ARN: $ROLE_ARN" | tee -a "$LOG_FILE"

# Set identity variables for cleanup
IDENTITY_TYPE="role"
IDENTITY_NAME="$ROLE_NAME"

# Attach the policy to the role
ATTACH_OUTPUT=$(log_cmd "aws iam attach-role-policy --role-name $ROLE_NAME --
policy-arn $POLICY_ARN")
ATTACH_STATUS=$?
check_error "$ATTACH_OUTPUT" $ATTACH_STATUS "Failed to attach policy to role
 $ROLE_NAME"

echo "Policy attached to role: $ROLE_NAME" | tee -a "$LOG_FILE"

# Step 3: Create Systems Manager configuration using Host Management
echo ""
echo "Step 3: Creating Systems Manager configuration..."

# Generate a random identifier for the configuration name
CONFIG_NAME="SSMSetup-$(openssl rand -hex 4)"

# Create configuration file for Systems Manager setup using Host Management
# Added both required parameters for single account deployment based on
 CloudFormation documentation
cat > ssm-config.json << EOF
[
```

```
  {
    "Type": "AWSQuickSetupType-SSMHostMgmt",
    "LocalDeploymentAdministrationRoleArn": "$ROLE_ARN",
    "LocalDeploymentExecutionRoleName": "$ROLE_NAME",
    "Parameters": {
      "TargetAccounts": "$ACCOUNT_ID",
      "TargetRegions": "$CURRENT_REGION"
    }
  }
]
EOF

echo "Configuration file created:" | tee -a "$LOG_FILE"
cat ssm-config.json | tee -a "$LOG_FILE"

# Create the configuration manager
CONFIG_OUTPUT=$(log_cmd "aws ssm-quicksetup create-configuration-manager --name
 \"$CONFIG_NAME\" --configuration-definitions file://ssm-config.json --region
 $CURRENT_REGION")
CONFIG_STATUS=$?
check_error "$CONFIG_OUTPUT" $CONFIG_STATUS "Failed to create Systems Manager
 configuration"

# Extract the manager ARN
MANAGER_ARN=$(echo "$CONFIG_OUTPUT" | grep -o 'arn:aws:ssm-quicksetup:[^"]*')
if [[ -z "$MANAGER_ARN" ]]; then
    echo "ERROR: Failed to extract manager ARN" | tee -a "$LOG_FILE"
    exit 1
fi

# Track the created configuration manager
track_resource "SSM_CONFIG_MANAGER" "$MANAGER_ARN"

echo "Created Systems Manager configuration: $MANAGER_ARN" | tee -a "$LOG_FILE"

# Step 4: Verify the setup
echo ""
echo "Step 4: Verifying the setup..."

# Wait for the configuration to be fully deployed
echo "Waiting for the configuration to be deployed (this may take a few
 minutes)..."
sleep 30
```

```
# Check the configuration manager status
VERIFY_OUTPUT=$(log_cmd "aws ssm-quicksetup get-configuration-manager --manager-
arn $MANAGER_ARN --region $CURRENT_REGION")
VERIFY_STATUS=$?
check_error "$VERIFY_OUTPUT" $VERIFY_STATUS "Failed to verify configuration
 manager"

echo "Systems Manager setup completed successfully!" | tee -a "$LOG_FILE"

# List the created resources
echo ""
echo "==========================================="
echo "CREATED RESOURCES"
echo "==========================================="
for resource in "${CREATED_RESOURCES[@]+"${CREATED_RESOURCES[@]}"}"; do
    echo "$resource"
done

# Prompt for cleanup
echo ""
echo "==========================================="
echo "CLEANUP CONFIRMATION"
echo "==========================================="
echo "Do you want to clean up all created resources? (y/n): "
CLEANUP_CHOICE="y"

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    echo "Cleaning up resources..." | tee -a "$LOG_FILE"
    cleanup_resources
    echo "Cleanup completed." | tee -a "$LOG_FILE"
else
    echo "Resources will not be cleaned up. You can manually clean them up
 later." | tee -a "$LOG_FILE"
fi

echo ""
echo "Script execution completed. See $LOG_FILE for details."

# Clean up temporary files
rm -f ssm-onboarding-policy.json trust-policy.json ssm-config.json 2>/dev/null ||
 true
```

- For API details, see the following topics in *AWS CLI Command Reference*.

- AttachRolePolicy
- CreateConfigurationManager
- CreatePolicy
- CreateRole
- DeleteConfigurationManager
- DeletePolicy
- DeleteRole
- DetachRolePolicy
- GetCallerIdentity
- GetConfigurationManager

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Security in IAM and AWS STS

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Identity and Access Management (IAM), see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS). The following topics show you how to configure IAM and AWS STS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your IAM resources.

Contents

- AWS security credentials
- AWS security audit guidelines
- Data protection in AWS Identity and Access Management
- Logging and monitoring in AWS Identity and Access Management
- Compliance validation for AWS Identity and Access Management
- Resilience in AWS Identity and Access Management
- Infrastructure security in AWS Identity and Access Management
- Configuration and vulnerability analysis in AWS Identity and Access Management
- AWS managed policies for AWS Identity and Access Management, IAM Access Analyzer, and account access manager

- Security features outside IAM

AWS security credentials

When you interact with AWS, you specify your AWS *security credentials* to verify who you are and whether you have permission to access the resources that you are requesting. AWS uses the security credentials to authenticate and authorize your requests.

For example, if you want to download a protected file from an Amazon Simple Storage Service (Amazon S3) bucket, your credentials must allow that access. If your credentials don't show you are authorized to download the file, AWS denies your request. However, your AWS security credentials aren't required for you to download a file in an Amazon S3 bucket that is publicly shared.

There are different types of users in AWS, each with their own security credentials:

- **Account owner (root user)** — The user who created the AWS account and has full access.
- **AWS IAM Identity Center users** — Users managed in AWS IAM Identity Center.
- **Federated principals** — Users from external identity providers who are granted temporary access to AWS through federation. For more information about federated identities, see Identity providers and federation into AWS.
- **IAM users** — Individual users created within the AWS Identity and Access Management (IAM) service.

Users have either long-term or temporary security credentials. Root user, IAM user, and access keys have long-term security credentials that do not expire. To protect long-term credentials have processes in place to manage access keys, change passwords, and enable MFA.

To simplify managing root user credentials across member accounts in AWS Organizations, you can centrally secure the root user credentials of your AWS accounts managed using AWS Organizations. Centrally manage root access for member accounts lets you centrally remove and prevent long-term root user credential recovery, preventing unintended root access at scale.

IAM roles, users in AWS IAM Identity Center, and AWS STS federated user principals have temporary security credentials. Temporary security credentials expire after a defined period of time or when the user ends their session. Temporary credentials work almost identically to long-term credentials, with the following differences:

- Temporary security credentials are *short-term*, as the name implies. They can be configured to last for anywhere from a few minutes to several hours. After the credentials expire, AWS no longer recognizes them or allows any kind of access from API requests made with them.
- Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested. When (or even before) the temporary security credentials expire, the user can request new credentials, as long as the user requesting them still has permissions to do so.

As a result, temporary credentials have the following advantages over long-term credentials:

- You do not have to distribute or embed long-term AWS security credentials with an application.
- You can provide access to your AWS resources to users without having to define an AWS identity for them. Temporary credentials are the basis for roles and identity federation.
- The temporary security credentials have a limited lifetime, so you do not have to update them or explicitly revoke them when they're no longer needed. After temporary security credentials expire, they cannot be reused. You can specify how long the credentials are valid, up to a maximum limit.

Security considerations

We recommend that you consider the following information when determining the security provisions for your AWS account:

- When you create an AWS account, we create the account root user. The credentials of the root user (account owner) allow full access to all resources in the account. The first task you perform with the root user is to grant another user administrative permissions to your AWS account so that you minimize the usage of the root user.
- Multi-factor authentication (MFA) provides an extra level of security for users who can access your AWS account. For additional security, we recommend that you require MFA on the AWS account root user credentials and all IAM users. For more information, see AWS Multi-factor authentication in IAM.
- AWS requires different types of security credentials, depending on how you access AWS and what type of AWS user you are. For example, you use sign-in credentials for the AWS Management Console while you use access keys to make programmatic calls to AWS. For help determining your user type and sign-in page, see What is AWS Sign-In in the *AWS Sign-In User Guide*.

- You can't use IAM policies to deny the root user access to resources explicitly. You can only use an AWS Organizations service control policy (SCP) to limit the permissions of the root user.
- If you forget or lose your root user password, you must have access to the email address associated with your account in order to reset it.
- If you lose your root user access keys, you must be able to sign in to your account as the root user to create new ones.
- Do not use the root user for your everyday tasks. Use it to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials.
- Security credentials are account-specific. If you have access to multiple AWS accounts, you have separate credentials for each account.
- Policies determine what actions a user, role, or member of a user group can perform, on which AWS resources, and under what conditions. Using policies you can securely control access to AWS services and resources in your AWS account. If you must modify or revoke permissions in response to a security event, you delete or modify the policies instead of making changes directly to the identity.
- Be sure to save the sign-in credentials for your *Emergency Access* IAM user and any access keys you created for programmatic access in a secure location. If you lose your access keys, you must sign in to your account to create new ones.
- We strongly recommend that you use temporary credentials provided by IAM roles and federated principals instead of the long-term credentials provided by IAM users and access keys.

Programmatic access with AWS security credentials

We recommend using short-term access keys when possible to make programmatic calls to AWS or to use the AWS Command Line Interface or AWS Tools for PowerShell. However, you can also use long-term AWS access keys for these purposes.

When you create a long-term access key, you create the access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`) as a set. The secret access key is available for download only when you create it. If you don't download your secret access key or if you lose it, you must create a new one.

In many scenarios, you don't need long-term access keys that never expire (as you have when you create access keys for an IAM user). Instead, you can create IAM roles and generate temporary security credentials. Temporary security credentials include an access key ID and a secret access key,

but they also include a security token that indicates when the credentials expire. After they expire, they're no longer valid. For more information, see Alternatives to long-term access keys

Access key IDs beginning with `AKIA` are long-term access keys for an IAM user or an AWS account root user. Access key IDs beginning with `ASIA` are temporary credentials access keys that you create using AWS STS operations.

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	**To**	**By**
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*.

Which user needs programmatic access?	To	By
		• For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Alternatives to long-term access keys

For many common use cases, there are alternatives to long-term access keys. To improve your account security, consider the following.

- **Don't embed long-term access keys and secret access keys in your application code or in a code repository –** Instead, use AWS Secrets Manager, or other secrets management solution, so you don't have to hardcode keys in plaintext. The application or client can then retrieve secrets when needed. For more information, see What is AWS Secrets Manager? in the *AWS Secrets Manager User Guide*.

- **Use IAM roles to generate temporary security credentials whenever possible –** Always use mechanisms to issue temporary security credentials when possible, rather than long-term access keys. Temporary security credentials are more secure because they are not stored with the user but are generated dynamically and provided to the user when requested. Temporary security credentials have a limited lifetime so you don't have to manage or update them. Mechanisms that provide temporary access keys include IAM roles or the authentication of an IAM Identity Center user. For machines that run outside of AWS you can use AWS Identity and Access Management Roles Anywhere.
- **Use alternatives to long-term access keys for the AWS Command Line Interface (AWS CLI) or the `aws-shell` –** Alternatives include the following.
  - **Login for AWS local development with console credentials**. You can use the AWS CLI Version 2 and the `aws login` command to generate short-term credentials to run AWS CLI commands using your console credentials. To learn more, see Login for AWS local development in the *AWS Command Line Interface User Guide*.
  - **AWS CloudShell** is a browser-based, pre-authenticated shell that you can launch directly from the AWS Management Console. You can run AWS CLI commands against AWS services through your preferred shell (Bash, Powershell, or Z shell). When you do this, you don't need to download or install command line tools. For more information, see What is AWS CloudShell? in the *AWS CloudShell User Guide*.
- **Don't create long-term access keys for human users who need access to applications or AWS services –** IAM Identity Center can generate temporary access credentials for your external IdP users to access AWS services. This eliminates the need to create and manage long-term credentials in IAM. In IAM Identity Center, create an IAM Identity Center permission set that grants the external IdP users access. Then assign a group from IAM Identity Center to the permission set in the selected AWS accounts. For more information, see What is AWS IAM Identity Center, Connect to your external identity provider, and Permission sets in the *AWS IAM Identity Center User Guide*.
- **Don't store long-term access keys within an AWS compute service –** Instead, assign an IAM role to compute resources. This automatically supplies temporary credentials to grant access.

For example, when you create an instance profile that is attached to an Amazon EC2 instance, you can assign an AWS role to the instance and make it available to all of its applications. An instance profile contains the role and enables programs that are running on the Amazon EC2 instance to get temporary credentials. To learn more, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances.

AWS security audit guidelines

Periodically audit your security configuration to make sure it meets your current business needs. An audit gives you an opportunity to remove unneeded IAM users, roles, groups, and policies, and to make sure that your users and software don't have excessive permissions.

Following are guidelines for systematically reviewing and monitoring your AWS resources for security best practices.

> **Tip**
>
> You can monitor your usage of IAM as it relates to security best practices by using AWS Security Hub CSPM. Security Hub CSPM uses security controls to evaluate resource configurations and security standards to help you comply with various compliance frameworks. For more information about using Security Hub CSPM to evaluate IAM resources, see AWS Identity and Access Management controls in the AWS Security Hub CSPM User Guide.

Contents

- When to perform a security audit
- Guidelines for auditing
- Review your AWS account credentials
- Review your IAM users
- Review your IAM groups
- Review your IAM roles
- Review your IAM providers for SAML and OpenID Connect (OIDC)
- Review Your mobile apps

- Tips for reviewing IAM policies

When to perform a security audit

Audit your security configuration in the following situations:

- On a periodic basis. Perform the steps described in this document at regular intervals as a best practice for security.
- If there are changes in your organization, such as people leaving.
- If you have stopped using one or more individual AWS services to verify that you have removed permissions that users in your account no longer need.
- If you've added or removed software in your accounts, such as applications on Amazon EC2 instances, OpsWorks stacks, CloudFormation templates, etc.
- If you suspect that an unauthorized person might have accessed your account.

Guidelines for auditing

As you review your account's security configuration, follow these guidelines:

- **Be thorough**. Look at all aspects of your security configuration, including those that are seldom used.
- **Don't assume**. If you are unfamiliar with some aspect of your security configuration (for example, the reasoning behind a particular policy or the existence of a role), investigate the business need until you understand the potential risk.
- **Keep things simple**. To make auditing (and management) easier, use IAM groups, IAM roles, consistent naming schemes, and straightforward policies.

Review your AWS account credentials

Take these steps when you audit your AWS account credentials:

1. If you have access keys for your root user that you're not using, you can remove them. We strongly recommend that you don't use root access keys for everyday work with AWS, and that instead you use users with temporary credentials, such users in AWS IAM Identity Center.
2. If you require access keys for your account, make sure you update them when needed.

Review your IAM users

Take these steps when you audit your existing IAM users:

1. List your users and then delete users that aren't needed.
2. Remove users from groups that they don't require access to.
3. Review the policies attached to the groups the user is in. See Tips for reviewing IAM policies.
4. Delete security credentials that the user doesn't need or that might have been exposed. For example, an IAM user used for an application doesn't need a password (which is necessary only to sign in to AWS websites). Similarly, if a user doesn't use access keys, there's no reason for the user to have one. For more information, see Managing Passwords for IAM users and Managing Access Keys for IAM users.

   You can generate and download a credential report that lists all IAM users in your account and the status of their various credentials, including passwords, access keys, and MFA devices. For passwords and access keys, the credential report shows the date and time when the password or access key was last used. Consider removing credentials that haven't been used recently from your account. (Don't remove your Emergency Access user.) For more information, see Getting Credential Reports for your AWS Account.
5. Update passwords and access keys when needed for use cases that require long-term credentials. For more information, see Managing Passwords for IAM Users and Managing Access Keys for IAM users.
6. As a best practice, require human users to use federation with an identity provider to access AWS using temporary credentials. If possible, transition from IAM users to federated principals, such as users in IAM Identity Center. Retain the minimum number of IAM users needed for your applications.

Review your IAM groups

Take these steps when you audit your IAM groups:

1. List your groups and then delete groups that you aren't using.
2. Review users in each group and remove users that don't belong.
3. Review the policies attached to the group. See Tips for reviewing IAM policies.

Review your IAM roles

Take these steps when you audit your IAM roles:

1. List your roles and then delete roles that you aren't using.
2. Review the role's trust policy. Make sure that you know who the principal is and that you understand why that account or user needs to be able to assume the role.
3. Review the access policy for the role to be sure that it grants suitable permissions to whoever assumes the role—see Tips for reviewing IAM policies.

Review your IAM providers for SAML and OpenID Connect (OIDC)

If you have created an IAM entity for establishing trust with a SAML or OIDC identity provider (IdP), take these steps:

1. Delete unused providers.
2. Download and review the AWS metadata documents for each SAML IdP and make sure the documents reflect your current business needs.
3. Get the latest metadata documents from the SAML IdPs and update the provider in IAM.

Review Your mobile apps

If you have created a mobile app that makes requests to AWS, take these steps:

1. Make sure that the mobile app doesn't contain embedded access keys, even if they're in encrypted storage.
2. Get temporary credentials for the app by using APIs designed for that purpose.

> **Note**
>
> We recommend that you use Amazon Cognito to manage user identity in your app. This service lets you authenticate users using Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC)–compatible identity provider. For more information, see Amazon Cognito identity pools in the *Amazon Cognito Developer Guide*.

Tips for reviewing IAM policies

Policies are powerful and subtle, so it's important to study and understand the permissions granted by each policy. Use the following guidelines when reviewing policies:

- Attach policies to groups or roles instead of to individual users. If an individual user has a policy, make sure you understand why that user needs the policy.
- Make sure that IAM users, groups, and roles have the permissions that they need and don't have any additional permissions.
- Use the IAM Policy Simulator to test policies attached to users or groups.
- Remember that a user's permissions are the result of all applicable policies— both identity-based policies (on users, groups,or roles) and resource-based policies (on resources such as Amazon S3 buckets, Amazon SQS queues, Amazon SNS topics, and AWS KMS keys). It's important to examine all the policies that apply to a user and to understand the complete set of permissions granted to an individual user.
- Be aware that allowing a user to create an IAM user, group, role, or policy and attach a policy to the principal entity is effectively granting that user all permissions to all resources in your account. Users who can create policies and attach them to a user, group, or role can grant themselves any permissions. In general, don't grant IAM permissions to users or roles whom you don't trust with full access to the resources in your account. When conducting your security audit confirm that the following IAM permissions are granted to trusted identities:
  - `iam:PutGroupPolicy`
  - `iam:PutRolePolicy`
  - `iam:PutUserPolicy`
  - `iam:CreatePolicy`
  - `iam:CreatePolicyVersion`
  - `iam:AttachGroupPolicy`
  - `iam:AttachRolePolicy`
  - `iam:AttachUserPolicy`
- Make sure policies don't grant permissions for services that you don't use. For example, if you use AWS managed policies, make sure the AWS managed policies that are in use in your account are for services that you actually use. To find out which AWS managed policies are in use in your account, use the IAM `GetAccountAuthorizationDetails` API (AWS CLI command: `aws iam get-account-authorization-details`).

- If the policy grants a user permission to launch an Amazon EC2 instance, it might also allow the `iam:PassRole` action, but if so it should explicitly list the roles that the user can pass to the Amazon EC2 instance.
- Examine any values for the `Action` or `Resource` element that include `*`. When possible, grant `Allow` access to the individual actions and resources that users need. However, the following are reasons that it might be suitable to use `*` in a policy:
  - The policy is designed to grant administrative-level permissions.
  - The wildcard character is used for a set of similar actions (for example, `Describe*`) as a convenience, and you are comfortable with the complete list of actions that are referenced in this way.
  - The wildcard character is used to indicate a class of resources or a resource path (for example, `arn:aws:iam::account-id:users/division_abc/*`), and you are comfortable granting access to all the resources in that class or path.
  - A service action doesn't support resource-level permissions, and the only choice for a resource is `*`.
- Examine policy names to make sure they reflect the policy's function. For example, although a policy might have a name that includes "read only," the policy might actually grant write or change permissions.

For more information about planning for your security audit, see Best Practices for Security, Identity, and Compliance in the *AWS Architecture Center*.

Data protection in AWS Identity and Access Management

The AWS shared responsibility model applies to data protection in AWS Identity and Access Management. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM).

That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with IAM or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption in IAM and AWS STS

Data encryption typically falls into two categories: encryption at rest and encryption in transit.

Encryption at rest

Data that is collected and stored by IAM is encrypted at rest.

- **IAM** – Data collected and stored within IAM includes IP addresses, customer account metadata, and customer identifying data that includes passwords. Customer account metadata and customer identifying data are encrypted at rest using AES 256 or is hashed using SHA 256.
- **AWS STS** – AWS STS does not collect customer content except for service logs that log successful, erroneous, and faulty requests to the service.

Encryption in transit

Customer identifying data, including passwords, is encrypted in transit using TLS 1.2 and 1.3. All AWS STS endpoints support HTTPS for encrypting data in transit. For a list of AWS STS endpoints, see AWS STS Regions and endpoints.

Key management in IAM and AWS STS

You can't manage encryption keys using IAM or AWS STS. For more information about encryption keys, see What is AWS KMS? in the AWS Key Management Service Developer Guide

Internetwork traffic privacy in IAM and AWS STS

Requests to IAM must be made using Transport Layer Security protocol (TLS). You can secure connections to the AWS STS service by using VPC endpoints. To learn more, see Interface VPC endpoints.

Logging and monitoring in AWS Identity and Access Management

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Identity and Access Management (IAM), AWS Security Token Service (AWS STS) and your other AWS solutions. AWS provides several tools for monitoring your AWS resources and responding to potential incidents:

- *AWS CloudTrail* captures all API calls for IAM and AWS STS as events, including calls from the console and API calls. To learn more about using CloudTrail with IAM and AWS STS, see Logging IAM and AWS STS API calls with AWS CloudTrail. For more information about CloudTrail, see the AWS CloudTrail User Guide.
- *AWS Identity and Access Management and Access Analyzer* helps you identify the resources in your organization and accounts, such as Amazon S3 buckets or IAM roles, that are shared with an external entity. This helps you identify unintended access to your resources and data, which is a security risk. To learn more, see What is IAM Access Analyzer?
- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify.

For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.

- *Amazon CloudWatch Logs* helps you monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.

For additional resources and security best practices for IAM, see Security best practices and use cases in AWS Identity and Access Management.

Topics

- Logging IAM and AWS STS API calls with AWS CloudTrail
- Track privileged tasks in AWS CloudTrail

Logging IAM and AWS STS API calls with AWS CloudTrail

IAM and AWS STS are integrated with AWS CloudTrail, a service that provides a record of actions taken by an IAM user or role. CloudTrail captures all API calls for IAM and AWS STS as events, including calls from the console and from API calls. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. You can use CloudTrail to get information about the request that was made to IAM or AWS STS. For example, you can view the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Topics

- IAM and AWS STS information in CloudTrail
- Logging IAM and AWS STS API requests
- Logging API requests to other AWS services
- Logging user sign-in events
- Logging sign-in events for temporary credentials
- Example IAM API events in CloudTrail log

- Example AWS STS API events in CloudTrail log
- Example sign-in events in CloudTrail log
- IAM role trust policy behavior

IAM and AWS STS information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in IAM or AWS STS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for IAM and AWS STS, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All IAM and AWS STS actions are logged by CloudTrail and are documented in the IAM API Reference and the AWS Security Token Service API Reference.

Logging IAM and AWS STS API requests

CloudTrail logs all authenticated API requests to IAM and AWS STS API operations. CloudTrail also logs non-authenticated requests to the AWS STS actions, `AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity`, and logs information provided by the identity provider. However, some non-authenticated AWS STS requests might not be logged because they do not meet the minimum expectation of being sufficiently valid to be trusted as a legitimate request. CloudTrail does not log denied AWS STS requests in the target account for cross-account role assumptions and privileged sessions initiated via the `AssumeRoot` API.

You can use the logged information to map calls made by an OIDC or SAML federated principal with an assumed role back to the originating external federated caller. In the case of `AssumeRole`, you can map calls back to the originating AWS service or to the account of the originating user. The `userIdentity` section of the JSON data in the CloudTrail log entry contains the information that you need to map the AssumeRole* request with a specific session principal. For more information, see CloudTrail userIdentity Element in the *AWS CloudTrail User Guide*.

AWS CloudTrail logs will contain MFA information when the IAM user sign in with MFA. If the IAM user assumes an IAM role, CloudTrail will also log `mfaAuthenticated: true` in the `sessionContext` attributes for actions performed using the assumed role. However, CloudTrail logging is separate from what IAM requires when API calls are made with the assumed role's credentials. For more information, see CloudTrail userIdentity Element.

For example, calls to the IAM `CreateUser`, `DeleteRole`, `ListGroups`, and other API operations are all logged by CloudTrail.

Examples for this type of log entry are presented later in this topic.

Logging API requests to other AWS services

Authenticated requests to other AWS service API operations are logged by CloudTrail, and these log entries contain information about who generated the request.

For example, assume that you made a request to list Amazon EC2 instances or create an AWS CodeDeploy deployment group. Details about the person or service that made the request are contained in the log entry for that request. This information helps you determine whether the request was made by the AWS account root user, an IAM user, a role, or another AWS service.

For more details about the user identity information in CloudTrail log entries, see userIdentity Element in the *AWS CloudTrail User Guide*.

Logging user sign-in events

CloudTrail logs sign-in events to the AWS Management Console, local development tools like AWS CLI and SDKs, the AWS discussion forums, and AWS Marketplace. CloudTrail logs successful and failed sign-in attempts for IAM users, SAML and OIDC federated principals, and AWS STS federated user principals.

To view sample CloudTrail events for successful and unsuccessful root user sign-ins, see Example event records for root users in the *AWS CloudTrail User Guide*.

As a security best practice, AWS does not log the entered IAM user name text when the sign-in failure is caused by *an incorrect user name*. The user name text is masked by the value `HIDDEN_DUE_TO_SECURITY_REASONS`. For an example of this, see Example sign-in failure event caused by incorrect user name, later in this topic. The user name text is obscured because such failures might be caused by user errors. Logging these errors could expose potentially sensitive information. For example:

- You accidentally type your password in the user name box.
- You choose the link for the sign-in page of one AWS account, but then type the account number for a different AWS account.
- You forget which account you are signing in to and accidentally type the account name of your personal email account, your bank sign-in identifier, or some other private ID.

Logging sign-in events for temporary credentials

When a principal requests temporary credentials, the principal type determines how CloudTrail logs the event. This can be complicated when a principal assumes a role in another account. There are multiple API calls to perform operations related to role cross-account operations. First, the principal calls an AWS STS API to retrieve the temporary credentials. That operation is logged in the calling account and the account where the AWS STS operation is performed. Then the principal then uses the role to perform other API calls in the assumed role's account.

You can use the `sts:SourceIdentity` condition key in the role trust policy to require users to specify an identity when they assume a role. For example, you can require that IAM users specify their own user name as their source identity. This can help you determine which user performed a specific action in AWS. For more information, see `sts:SourceIdentity`. You can also use `sts:RoleSessionName` to require users to specify a session name when they assume a role. This can help you differentiate between role sessions for a role that is used by different principals when you review AWS CloudTrail logs.

The following table shows how CloudTrail logs different user identity information for each of the AWS STS APIs that generate temporary credentials.

Principal type	STS API	User identity in CloudTrail log for caller's account	User identity in CloudTrail log for the assumed role's account	User identity in CloudTrail log for the role's subsequent API calls
AWS account root user credentials	GetSessionToken	Root user identity	Role owner account is same as calling account	Root user identity
AWS account root user credentials	AssumeRoot	Root user session	Account number and principal ID (if a user)	Root user session
IAM user	GetSessionToken	IAM user identity	Role owner account is same as calling account	IAM user identity
IAM user	GetFederationToken	IAM user identity	Role owner account is same as calling account	IAM user identity
IAM user	AssumeRole	IAM user identity	Account number and principal ID (if a user), or AWS service principal	Role identity only (no user)
Externally authenticated user	AssumeRoleWithSAML	n/a	SAML user identity	Role identity only (no user)
Externally authenticated user	AssumeRoleWithWebIdentity	n/a	OIDC/Web user identity	Role identity only (no user)

CloudTrail considers an action read-only if it does not have any mutating effect on a resource. When logging a read-only event, CloudTrail redacts the `responseElements` information in the log. When CloudTrail logs an event that is not read-only, the full `responseElements` is shown in the log entry. For the AWS STS APIs `AssumeRole`, `AssumeRoleWithSAML`, and `AssumeRoleWithWebIdentity`, even though they are logged as read-only, CloudTrail will include the full `responseElements` except `secretAccessKey` in the log for these APIs.

The following table shows how CloudTrail logs `responseElements` and `readOnly` information for each of the AWS STS APIs that generate temporary credentials.

STS API	Response elements information	Read-only
AssumeRole	Included	true
AssumeRoleWithSAML	Included	true
AssumeRoleWithWebIdentity	Included	true
AssumeRoot	Included	false
GetFederationToken	Included	false
GetSessionToken	Included	false

Example IAM API events in CloudTrail log

CloudTrail log files contain events that are formatted using JSON. An API event represents a single API request and includes information about the principal, the requested action, any parameters, and the date and time of the action.

Example IAM API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made for the IAM `GetUserPolicy` action.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
```

```
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:Role-Session-Name",
        "arn": "arn:aws:sts::111122223333:assumed-role/Role-Name/Role-Session-Name",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2024-09-09T17:50:16Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-09-09T17:51:44Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "GetUserPolicy",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.101",
    "userAgent": "aws-cli/1.16.96 Python/2.7.8 Linux/10 botocore/1.12.86",
    "requestParameters": {
        "userName": "ExampleIAMUserName",
        "policyName": "ExamplePoliccyName"
    },
    "responseElements": null,
    "requestID": "9EXAMPLE-0c68-11e4-a24e-d5e16EXAMPLE",
    "eventID": "cEXAMPLE-127e-4632-980d-505a4EXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "iam.amazonaws.com"
    }
}
```

From this event information, you can determine that the request was made to get a user policy named `ReadOnlyAccess-JaneDoe-201407151307` for user `JaneDoe`, as specified in the `requestParameters` element. You can also see that the request was made by an IAM user named `JaneDoe` on July 15, 2014 at 9:40 PM (UTC). In this case, the request originated in the AWS Management Console, as you can tell from the `userAgent` element.

Example AWS STS API events in CloudTrail log

CloudTrail log files contain events that are formatted using JSON. An API event represents a single API request and includes information about the principal, the requested action, any parameters, and the date and time of the action.

Example cross-account AWS STS API events in CloudTrail log files

The IAM user named `John` in account 777788889999 calls the AWS STS AssumeRole action to assume the role `EC2-dev` in account 111122223333. The account administrator requires users to set a source identity equal to their user name when assuming the role. The user passes in the source identity value of `John`.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAQRSTUVWXYZEXAMPLE",
    "arn": "arn:aws:iam::777788889999:user/John",
    "accountId": "777788889999",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "John"
  },
  "eventTime": "2014-07-18T15:07:39Z",
  "eventSource": "sts.amazonaws.com",
  "eventName": "AssumeRole",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "aws-cli/1.11.10 Python/2.7.8
 Linux/3.2.45-0.6.wd.865.49.315.metal1.x86_64 botocore/1.4.67",
  "requestParameters": {
    "roleArn": "arn:aws:iam::111122223333:role/EC2-dev",
    "roleSessionName": "John-EC2-dev",
    "sourceIdentity": "John",
    "serialNumber": "arn:aws:iam::777788889999:mfa"
  },
```

```
  "responseElements": {
    "credentials": {
      "sessionToken": "<encoded session token blob>",
      "accessKeyId": "ASIAI44QH8DHBEXAMPLE",
      "expiration": "Jul 18, 2023, 4:07:39 PM"
      },
    "assumedRoleUser": {
      "assumedRoleId": "AIDAQRSTUVWXYZEXAMPLE:John-EC2-dev",
      "arn": "arn:aws:sts::111122223333:assumed-role/EC2-dev/John-EC2-dev"
    },
    "sourceIdentity": "John",
    "sessionTokenSize": 1428,
    "sessionTokenUtilization": 34
  },
  "resources": [
    {
      "ARN": "arn:aws:iam::111122223333:role/EC2-dev",
      "accountId": "111122223333",
      "type": "AWS::IAM::Role"
    }
  ],
  "requestID": "4EXAMPLE-0e8d-11e4-96e4-e55c0EXAMPLE",
  "sharedEventID": "bEXAMPLE-efea-4a70-b951-19a88EXAMPLE",
  "eventID": "dEXAMPLE-ac7f-466c-a608-4ac8dEXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

The second example shows the assumed role account's (111122223333) CloudTrail log entry for the same request.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AWSAccount",
    "principalId": "AIDAQRSTUVWXYZEXAMPLE",
    "accountId": "777788889999"
  },
  "eventTime": "2014-07-18T15:07:39Z",
  "eventSource": "sts.amazonaws.com",
  "eventName": "AssumeRole",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.101",
```

```
  "userAgent": "aws-cli/1.11.10 Python/2.7.8
 Linux/3.2.45-0.6.wd.865.49.315.metal1.x86_64 botocore/1.4.67",
  "requestParameters": {
    "roleArn": "arn:aws:iam::111122223333:role/EC2-dev",
    "roleSessionName": "John-EC2-dev",
    "sourceIdentity": "John",
    "serialNumber": "arn:aws:iam::777788889999:mfa"
  },
  "responseElements": {
    "credentials": {
      "sessionToken": "<encoded session token blob>",
      "accessKeyId": "ASIAI44QH8DHBEXAMPLE",
      "expiration": "Jul 18, 2014, 4:07:39 PM"
    },
    "assumedRoleUser": {
      "assumedRoleId": "AIDAQRSTUVWXYZEXAMPLE:John-EC2-dev",
      "arn": "arn:aws:sts::111122223333:assumed-role/EC2-dev/John-EC2-dev"
      },
    "sourceIdentity": "John",
    "sessionTokenSize": 1428,
    "sessionTokenUtilization": 34
  },
  "requestID": "4EXAMPLE-0e8d-11e4-96e4-e55c0EXAMPLE",
  "sharedEventID": "bEXAMPLE-efea-4a70-b951-19a88EXAMPLE",
  "eventID": "dEXAMPLE-ac7f-466c-a608-4ac8dEXAMPLE"
}
```

Example AWS STS role chaining API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made by John Doe in account 111111111111. John previously used his `John` user to assume the `JohnRole1` role. For this request, he uses the credentials from that role to assume the `JohnRole2` role. This is known as role chaining. The source identity that he set when he assumed the `John1` role persists in the request to assume `JohnRole2`. If John tries to set a different source identity when assuming the role, the request is denied. John passes two session tags into the request. He sets those two tags as transitive. The request inherits the `Department` tag as transitive because John set it as transitive when he assumed `JohnRole1`. For more information about source identity, see Monitor and control actions taken with assumed roles. For more information about transitive keys in role chains, see Chaining roles with session tags.

```
{
    "eventVersion": "1.05",
```

```
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIN5ATK5U7KEXAMPLE:JohnRole1",
        "arn": "arn:aws:sts::111111111111:assumed-role/John/JohnRole1",
        "accountId": "111111111111",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2019-10-02T21:50:54Z"
            },
                "sessionIssuer": {
                "type": "Role",
               "principalId": "AROAIN5ATK5U7KEXAMPLE",
                "arn": "arn:aws:iam::111111111111:role/JohnRole1",
                "accountId": "111111111111",
                "userName": "John"
            },
            "sourceIdentity": "John"
        }
    },
    "eventTime": "2019-10-02T22:12:29Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRole",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "123.145.67.89",
    "userAgent": "aws-cli/1.16.248 Python/3.4.7
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.12.239",
    "requestParameters": {
        "incomingTransitiveTags": {
            "Department": "Engineering"
        },
        "tags": [
            {
                "value": "johndoe@example.com",
                "key": "Email"
            },
            {
                "value": "12345",
                "key": "CostCenter"
            }
        ],
        "roleArn": "arn:aws:iam::111111111111:role/JohnRole2",
        "roleSessionName": "Role2WithTags",
```

```
        "sourceIdentity": "John",
        "transitiveTagKeys": [
            "Email",
            "CostCenter"
        ],
        "durationSeconds": 3600
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAI44QH8DHBEXAMPLE",
            "expiration": "Oct 2, 2019, 11:12:29 PM",
            "sessionToken": "AgoJb3JpZ2luX2VjEB4aCXVzLXdlc3QtMSJHMEXAMPLETOKEN
+//rJb8Lo30mFc5MlhFCEbubZvEj0wHB/mDMwIgSEe9gk/Zjr09tZV7F1HDTMhmEXAMPLETOKEN/iEJ/
rkqngII9//////////
ARABGgw0MjgzMDc4NjM5NjYiDLZjZFKwP4qxQG5sFCryASO4UPz5qE97wPPH1eLMvs7CgSDBSWfonmRTCfokm2FN1+hWUdQ
+C+WKFZb701eiv9J5La2EXAMPLETOKEN/c7S5Iro1WUJ0q3Cxuo/8HUoSxVhQHM7zF7mWWLhXLEQ52ivL
+F6q5dpXu4aTFedpMfnJa8JtkWwG9x1Axj0Ypy2ok8v5unpQGWych1vwdvj6ez1Dm8Xg1+qIzXILiEXAMPLETOKEN/
vQGqu8H+nxp3kabcrtOvTFTvxX6vsc8OGwUfHhzAfYGEXAMPLETOKEN/
L6v1yMM3B1OwFOrQBno1HEjf1oNI8RnQiMNFdUOtwYj7HUZIOCZmjfN8PPHq77N7GJl9lzvIZKQA0Owcjg
+mc78zHCj8y0siY8C96paEXAMPLETOKEN/
E3cpksxWdgs91HRzJWScjN2+r2LTGjYhyPqcmFzzo2mCE7mBNEXAMPLETOKEN/oJy
+2o83YNW5tOiDmczgDzJZ4UKR84yGYOMfSnF4XcEJrDgAJ3OJFwmTcTQICAlSwLEXAMPLETOKEN"
        },
        "assumedRoleUser": {
            "assumedRoleId": "AROAIFR7WHDTSOYQYHFUE:Role2WithTags",
            "arn": "arn:aws:sts::111111111111:assumed-role/test-role/Role2WithTags"
        },
        "sourceIdentity": "John",
        "sessionTokenSize": 1652,
        "sessionTokenUtilization": 40
    },
    "requestID": "b96b0e4e-e561-11e9-8b3f-7b396EXAMPLE",
    "eventID": "1917948f-3042-46ec-98e2-62865EXAMPLE",
    "resources": [
        {
            "ARN": "arn:aws:iam::111111111111:role/JohnRole2",
            "accountId": "111111111111",
            "type": "AWS::IAM::Role"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111111111111"
}
```

Example AWS service AWS STS API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made by an AWS service calling another service API using permissions from a service role. It shows the CloudTrail log entry for the request made in account 777788889999.


```
{
  "eventVersion": "1.04",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAQRSTUVWXYZEXAMPLE:devdsk",
    "arn": "arn:aws:sts::777788889999:assumed-role/AssumeNothing/devdsk",
    "accountId": "777788889999",
    "accessKeyId": "ASIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2016-11-14T17:25:26Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAQRSTUVWXYZEXAMPLE",
        "arn": "arn:aws:iam::777788889999:role/AssumeNothing",
        "accountId": "777788889999",
        "userName": "AssumeNothing"
      }
    }
  },
  "eventTime": "2016-11-14T17:25:45Z",
  "eventSource": "s3.amazonaws.com",
  "eventName": "DeleteBucket",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.1",
  "userAgent": "[aws-cli/1.11.10 Python/2.7.8
 Linux/3.2.45-0.6.wd.865.49.315.metal1.x86_64 botocore/1.4.67]",
  "requestParameters": {
    "bucketName": "amzn-s3-demo-bucket"
  },
  "responseElements": null,
  "requestID": "EXAMPLE463D56D4C",
  "eventID": "dEXAMPLE-265a-41e0-9352-4401bEXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "777788889999"
```

```
}
```

Example SAML AWS STS API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made for the AWS STS AssumeRoleWithSAML action. The request includes the SAML attributes `CostCenter` and `Project` that are passed through the SAML assertion as session tags. Those tags are set as transitive so that they persist in role chaining scenarios. The request includes the optional API parameter `DurationSeconds`, represented as `durationSeconds` in the CloudTrail log, and is set to `1800` seconds. The request also includes the SAML attribute `sourceIdentity`, which is passed in the SAML assertion. If someone uses the resulting role session credentials to assume another role, this source identity persists.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "SAMLUser",
        "principalId": "SampleUkh1i4+ExamplexL/jEvs=:SamlExample",
        "userName": "SamlExample",
        "identityProvider": "bdGOnTesti4+ExamplexL/jEvs="
    },
    "eventTime": "2023-08-28T18:30:58Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRoleWithSAML",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "aws-internal/3 aws-sdk-java/1.12.479
 Linux/5.10.186-157.751.amzn2int.x86_64 OpenJDK_64-Bit_Server_VM/17.0.7+11 java/17.0.7
 kotlin/1.3.72 vendor/Amazon.com_Inc. cfg/retry-mode/standard",
    "requestParameters": {
        "sAMLAssertionID": "_c0046cEXAMPLEb9d4b8eEXAMPLE2619aEXAMPLE",
        "roleSessionName": "MyAssignedRoleSessionName",
        "sourceIdentity": "MySAMLUser",
        "principalTags": {
            "CostCenter": "987654",
            "Project": "Unicorn",
            "Department": "Engineering"
        },
        "transitiveTagKeys": [
            "CostCenter",
            "Project"
        ],
```

```
        "roleArn": "arn:aws:iam::444455556666:role/SAMLTestRoleShibboleth",
        "principalArn": "arn:aws:iam::444455556666:saml-provider/Shibboleth",
        "durationSeconds": 1800
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
            "sessionToken": "<encoded session token blob>",
            "expiration": "Aug 28, 2023, 7:00:58 PM"
        },
        "assumedRoleUser": {
            "assumedRoleId": "AROAD35QRSTUVWEXAMPLE:MyAssignedRoleSessionName",
            "arn": "arn:aws:sts::444455556666:assumed-role/SAMLTestRoleShibboleth/
MyAssignedRoleSessionName"
        },
        "packedPolicySize": 34,
        "sessionTokenSize": 1428,
        "sessionTokenUtilization": 34,
        "subject": "SamlExample",
        "subjectType": "transient",
        "issuer": "https://server.example.com/idp/shibboleth",
        "audience": "https://signin.aws.amazon.com/saml",
        "nameQualifier": "bdGOnTesti4+ExamplexL/jEvs=",
        "sourceIdentity": "MySAMLUser"
    },
    "requestID": "6EXAMPLE-e595-11e5-b2c7-c974fEXAMPLE",
    "eventID": "dEXAMPLE-265a-41e0-9352-4401bEXAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "444455556666",
            "type": "AWS::IAM::Role",
            "ARN": "arn:aws:iam::444455556666:role/SAMLTestRoleShibboleth"
        },
        {
            "accountId": "444455556666",
            "type": "AWS::IAM::SAMLProvider",
            "ARN": "arn:aws:iam::444455556666:saml-provider/test-saml-provider"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "444455556666",
    "eventCategory": "Management",
```

```
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "sts.us-east-2.amazonaws.com"
    }
}
```

Example OIDC AWS STS API event in CloudTrail log file

The following example shows a CloudTrail log entry for a request made for the AWS STS AssumeRoleWithWebIdentity action. The request includes the attributes `CostCenter` and `Project` that are passed through the OpenID Connect (OIDC) identity provider (IdP) token as session tags. Those tags are set as transitive so that they persist in role chaining. The request includes the `sourceIdentity` attribute from the identity provider token. If someone uses the resulting role session credentials to assume another role, this source identity persists.

The CloudTrail log entry also contains an `additionalEventData` field with an `identityProviderConnectionVerificationMethod` attribute. This attribute indicates the method AWS used to verify the connection with the OIDC provider. The attribute value will be either `IAMTrustStore` or `Thumbprint`. The `IAMTrustStore` value indicates that AWS successfully verified the connection with the OIDC IdP using our library of trusted root certificate authorities (CAs). The `Thumbprint` value indicates that AWS used a certificate thumbprint set in the IdP configuration to verify the OIDC IdP server certificate.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "WebIdentityUser",
    "principalId": "arn:aws:iam::444455556666:oidc-provider/<issuer url of OIDC
 provider>:<id of application>:<id of user>",
    "userName": "<id of user>",
    "identityProvider": "arn:aws:iam::444455556666:oidc-provider/<issuer url of OIDC
 provider>"
  },
  "eventTime": "2024-07-09T15:41:37Z",
  "eventSource": "sts.amazonaws.com",
  "eventName": "AssumeRoleWithWebIdentity",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "aws-cli/2.13.29 Python/3.11.6 Windows/10 exe/AMD64 prompt/off command/
sts.assume-role-with-web-identity",
```

```
  "requestParameters": {
    "roleArn": "arn:aws:iam::444455556666:role/FederatedWebIdentityRole",
    "roleSessionName": "<assigned role session name>",
    "sourceIdentity": "MyWebIdentityUser",
    "durationSeconds": 3600,
    "principalTags": {
      "CostCenter": "24680",
      "Project": "Pegasus"
    },
    "transitiveTagKeys": [
      "CostCenter",
      "Project"
    ]
  },
  "responseElements": {
    "credentials": {
      "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
      "sessionToken": "<encoded session token blob>",
      "expiration": "Jul 9, 2024, 4:41:37 PM"
    },
    "subjectFromWebIdentityToken": "<id of user>",
    "sourceIdentity": "MyWebIdentityUser",
    "assumedRoleUser": {
      "assumedRoleId": "AROA123456789EXAMPLE:<assigned role session name>",
      "arn": "arn:aws:sts::444455556666:assumed-role/FederatedWebIdentityRole/<assigned
role session name>"
    },
    "provider": "arn:aws:iam::444455556666:oidc-provider/<issuer url of OIDC
provider>",
    "audience": "<id of application>",
    "sessionTokenSize": 1536,
    "sessionTokenUtilization": 37
  },
  "additionalEventData": {
    "identityProviderConnectionVerificationMethod": "IAMTrustStore"
  },
  "requestID": "aEXAMPLE-0b26-40df-8973-c7012EXAMPLE",
  "eventID": "aEXAMPLE-ee29-4ac0-a0ed-3f5c5EXAMPLE",
  "readOnly": true,
  "resources": [
    {
      "accountId": "444455556666",
      "type": "AWS::IAM::Role",
      "ARN": "arn:aws:iam::444455556666:role/FederatedWebIdentityRole"
```

```
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "444455556666",
  "eventCategory": "Management",
  "tlsDetails": {
    "tlsVersion": "TLSv1.3",
    "cipherSuite": "TLS_AES_128_GCM_SHA256",
    "clientProvidedHostHeader": "sts.us-east-2.amazonaws.com"
  }
}
```

Example AWS STS API event using the global endpoint in CloudTrail log file

For requests to the AWS Security Token Service (AWS STS) global endpoint (`https://sts.amazonaws.com`), AWS STS includes additional AWS CloudTrail log fields: `endpointType` and `awsServingRegion`. These fields appear under the `additionalEventData` `RequestDetails` element to log the serving AWS Region and endpoint type being called. The `endpointType` field can have a value of `global` or `regional` to indicate the type of global endpoint that served the request. For more information about the AWS STS global endpoint changes, see AWS STS Regions and endpoints.

> **Note**
>
> AWS CloudTrail logs for requests made to the AWS STS global endpoint will be sent to the US East (N. Virginia) Region. CloudTrail logs for requests served by AWS STS Regional endpoints will continue to be logged to their respective Region in CloudTrail.

The following example shows a CloudTrail log entry for an AWS STS request made to the global endpoint (`https://sts.amazonaws.com`) that originated from the Europe (Stockholm) Region - eu-north-1. The `endpointType` field value of `global` indicates that the AWS STS request was served by the global endpoint in the Europe (Stockholm) Region.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:developer",
        "arn": "arn:aws:sts::777788889999:assumed-role/Admin/developer",
```

```
        "accountId": "777788889999",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::777788889999:role/Admin",
                "accountId": "777788889999",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2025-02-12T21:44:28Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-02-12T22:16:48Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRole",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/2.15.33 Python/3.11.8 Linux/5.10.233-204.894.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/sts.assume-role",
    "requestParameters": {
        "roleArn": "arn:aws:iam::777788889999:role/test-role",
        "roleSessionName": "test-global-assume-role"
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
            "sessionToken": "<encoded session token blob>",
            "expiration": "Feb 12, 2025, 11:16:48 PM"
        },
        "assumedRoleUser": {
            "assumedRoleId": "AROA987654321EXAMPLE:test-global-assume-role",
            "arn": "arn:aws:sts::777788889999:assumed-role/test-role/test-global-
assume-role"
        },
        "sessionTokenSize": 1096,
        "sessionTokenUtilization": 26
    },
    "additionalEventData": {
        "RequestDetails": {
```

```
            "awsServingRegion": "eu-north-1",
            "endpointType": "global"
        }
    },
    "requestID": "EXAMPLE7-2497-457a-9586-f21feEXAMPLE",
    "eventID": "EXAMPLEc-3d26-4c3a-9c94-722a9EXAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "777788889999",
            "type": "AWS::IAM::Role",
            "ARN": "arn:aws:iam::777788889999:role/test-role"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777788889999",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "sts-global.eu-north-1.amazonaws.com"
    }
}
```

For comparison, the following example shows a CloudTrail log entry for an AWS STS request made to the Regional endpoint (`https://sts.us-west-2.amazonaws.com`) that was served by the Regional endpoint in the Europe (Stockholm) Region - eu-north-1. The `endpointType` field value of `regional` indicates that the AWS STS request was served by the global endpoint in the Europe (Stockholm) Region.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:developer",
        "arn": "arn:aws:sts::777788889999:assumed-role/Admin/developer",
        "accountId": "777788889999",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
```

```
                "arn": "arn:aws:iam::777788889999:role/Admin",
                "accountId": "777788889999",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2025-02-12T21:44:28Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-02-12T22:16:30Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRole",
    "awsRegion": "eu-north-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/2.15.33 Python/3.11.8 Linux/5.10.233-204.894.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/sts.assume-role",
    "requestParameters": {
        "roleArn": "arn:aws:iam::777788889999:role/test-role",
        "roleSessionName": "test-global-assume-role"
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
            "sessionToken": "<encoded session token blob>",
            "expiration": "Feb 12, 2025, 11:16:30 PM"
        },
        "assumedRoleUser": {
            "assumedRoleId": "AROA987654321EXAMPLE:test-global-assume-role",
            "arn": "arn:aws:sts::777788889999:assumed-role/test-role/test-global-
assume-role"
        },
        "sessionTokenSize": 1096,
        "sessionTokenUtilization": 26
    },
    "additionalEventData": {
        "RequestDetails": {
            "endpointType": "regional",
            "awsServingRegion": "eu-north-1"
        }
    },
    "requestID": "EXAMPLEd-2116-4cd7-bd72-9f72fEXAMPLE",
    "eventID": "EXAMPLEd-219a-48ed-bc54-00e3cEXAMPLE",
```

```
    "readOnly": true,
    "resources": [
        {
            "accountId": "777788889999",
            "type": "AWS::IAM::Role",
            "ARN": "arn:aws:iam::777788889999:role/test-role"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777788889999",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "sts.eu-north-1.amazonaws.com"
    }
}
```

Example sign-in events in CloudTrail log

CloudTrail log files contain events that are formatted using JSON. A sign-in event represents a single sign-in request and includes information about the sign-in principal, the Region, and the date and time of the action.

Example sign-in success event in CloudTrail log file

The following example shows a CloudTrail log entry for a successful sign-in event.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn":"arn:aws:iam::111122223333:user/John",
    "accountId": "111122223333",
    "userName": "John"
  },
  "eventTime": "2014-07-16T15:49:27Z",
  "eventSource": "signin.amazonaws.com",
  "eventName": "ConsoleLogin",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.110",
```

```
  "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101
 Firefox/24.0",
  "requestParameters": null,
  "responseElements": {
    "ConsoleLogin": "Success"
  },
  "additionalEventData": {
    "MobileVersion": "No",
    "LoginTo": "https://console.aws.amazon.com/s3/  ",
    "MFAUsed": "No"
  },
  "eventID": "3fcfb182-98f8-4744-bd45-10a395ab61cb"
}
```


The following example shows a CloudTrail log entry for a successful authorization code request.


```
{
"eventVersion": "1.11",
"userIdentity": {
"type": "AssumedRole",
"principalId": "AROATJHQDX737YZP72NTF:thesjain-Isengard",
"arn": "arn:aws:sts::225989345271:assumed-role/Admin/thesjain-Isengard",
"accountId": "225989345271",
"sessionContext": {
"sessionIssuer": {
"type": "Role",
"principalId": "AROATJHQDX737YZP72NTF",
"arn": "arn:aws:iam::225989345271:role/Admin",
"accountId": "225989345271",
"userName": "Admin"
},
"attributes": {
"creationDate": "2025-11-17T22:50:14Z",
"mfaAuthenticated": "false"
}
}
},
"eventTime": "2025-11-17T22:51:32Z",
"eventSource": "signin.amazonaws.com",
"eventName": "AuthorizeOAuth2Access",
"awsRegion": "us-east-1",
"sourceIPAddress": "52.94.133.136",
```

```
"userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36",
"requestParameters": {
"scope": "openid",
"redirect_uri": "http://127.0.0.1:53037/oauth/callback",
"code_challenge_method": "SHA-256",
"client_id": "arn:aws:signin:::devtools/same-device"
},
"responseElements": null,
"additionalEventData": {
"success": "true",
"x-amzn-vpce-id": ""
},
"requestID": "e2854c76-1cba-4360-9fd1-5037b591466b",
"eventID": "59e1720d-3deb-44ff-933d-6828be2a860a",
"readOnly": true,
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "225989345271",
"eventCategory": "Management",
"tlsDetails": {
"tlsVersion": "TLSv1.3",
"cipherSuite": "TLS_AES_128_GCM_SHA256",
"clientProvidedHostHeader": "us-east-1.signin.aws.amazon.com"
}
}
```


The following example shows a CloudTrail log entry for a successful OAuth2 token creation request.


```
{
"eventVersion": "1.11",
"userIdentity": {
"type": "AssumedRole",
"principalId": "AROATJHQDX737YZP72NTF:jacobjoj-Isengard",
"arn": "arn:aws:sts::225989345271:assumed-role/Admin/jacobjoj-Isengard",
"accountId": "225989345271",
"sessionContext": {
"sessionIssuer": {
"type": "Role",
```

```
"principalId": "AROATJHQDX737YZP72NTF",
"arn": "arn:aws:iam::225989345271:role/Admin",
"accountId": "225989345271",
"userName": "Admin"
},
"attributes": {
"creationDate": "2025-11-18T20:38:10Z",
"mfaAuthenticated": "false"
}
}
},
"eventTime": "2025-11-18T20:38:44Z",
"eventSource": "signin.amazonaws.com",
"eventName": "CreateOAuth2Token",
"awsRegion": "us-east-1",
"sourceIPAddress": "15.248.6.6",
"userAgent": "aws-cli/2.32.0 md/awscrt#0.28.4 ua/2.1 os/macos#24.6.0 md/arch#arm64
 lang/python#3.13.9 md/pyimpl#CPython m/b,AA,Z,E cfg/retry-mode#standard md/
installer#exe sid/35033f4ca1bd md/prompt#off md/command#login",
"requestParameters": {
"client_id": "arn:aws:signin:::devtools/same-device"
},
"responseElements": null,
"additionalEventData": {
"success": "true",
"x-amzn-vpce-id": ""
},
"requestID": "94562943-c85b-4dc1-bf72-43b0fd42d6de",
"eventID": "0b338fac-6a10-4740-b34d-1bb6923e799e",
"readOnly": true,
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "225989345271",
"eventCategory": "Management",
"tlsDetails": {
"tlsVersion": "TLSv1.3",
"cipherSuite": "TLS_AES_128_GCM_SHA256",
"clientProvidedHostHeader": "us-east-1.signin.aws.amazon.com"
}
}
```

For more details about the information contained in CloudTrail log files, see CloudTrail Event Reference in the *AWS CloudTrail User Guide*.

Example sign-in failure event in CloudTrail log file

The following example shows a CloudTrail log entry for a failed sign-in event.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn":"arn:aws:iam::111122223333:user/JaneDoe",
    "accountId": "111122223333",
    "userName": "JaneDoe"
  },
  "eventTime": "2014-07-08T17:35:27Z",
  "eventSource": "signin.amazonaws.com",
  "eventName": "ConsoleLogin",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.100",
  "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101
 Firefox/24.0",
  "errorMessage": "Failed authentication",
  "requestParameters": null,
  "responseElements": {
    "ConsoleLogin": "Failure"
  },
  "additionalEventData": {
    "MobileVersion": "No",
    "LoginTo": "https://console.aws.amazon.com/sns",
    "MFAUsed": "No"
  },
  "eventID": "11ea990b-4678-4bcd-8fbe-62509088b7cf"
}
```

From this information, you can determine that the sign-in attempt was made by an IAM user named `JaneDoe`, as shown in the `userIdentity` element. You can also see that the sign-in attempt failed, as shown in the `responseElements` element. You can see that `JaneDoe` tried to sign in to the Amazon SNS console at 5:35 PM (UTC) on July 8, 2014.

Example sign-in failure event caused by incorrect user name

The following example shows a CloudTrail log entry for an unsuccessful sign-in event caused by the user entering an incorrect user name. AWS masks the `userName` text with `HIDDEN_DUE_TO_SECURITY_REASONS` to help prevent exposing potentially sensitive information.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "accountId": "123456789012",
    "accessKeyId": "",
    "userName": "HIDDEN_DUE_TO_SECURITY_REASONS"
  },
  "eventTime": "2015-03-31T22:20:42Z",
  "eventSource": "signin.amazonaws.com",
  "eventName": "ConsoleLogin",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.101",
  "userAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101
 Firefox/24.0",
  "errorMessage": "No username found in supplied account",
  "requestParameters": null,
  "responseElements": {
    "ConsoleLogin": "Failure"
  },
  "additionalEventData": {
    "LoginTo": "https://console.aws.amazon.com/console/home?state=hashArgs
%23&isauthcode=true",
    "MobileVersion": "No",
    "MFAUsed": "No"
  },
  "eventID": "a7654656-0417-45c6-9386-ea8231385051",
  "eventType": "AwsConsoleSignin",
  "recipientAccountId": "123456789012"
}
```

IAM role trust policy behavior

On September 21st, 2022, AWS made changes to IAM role trust policy behavior to require explicit allows in a role trust policy when a role assumes itself. IAM roles in the legacy behavior allow list have an additionalEventData field present for explicitTrustGrant for `AssumeRole` events.

The value of `explicitTrustGrant` is false when a role on the legacy allow list assumes itself using the legacy behavior. When a role on the legacy allow list assumes itself but the role trust policy behavior has been updated to explicitly allow the role to assume itself, the value of `explicitTrustGrant` is true.

Only a very small number of IAM roles are on the allow list for the legacy behavior, and this field is only present in CloudTrail logs for these roles when they assume themselves. In most cases, it is not necessary for an IAM role to assume itself. AWS recommends updating your processes, code, or configurations to remove this behavior or updating your role trust policies to explicitly allow for this behavior. For more information, see Announcing an update to IAM role trust policy behavior.

Track privileged tasks in AWS CloudTrail

The AWS Organizations management account or a delegated administrator account for IAM can perform some root user tasks on member accounts using short-term root access. Short-term privileged sessions give you temporary credentials that you can scope to take privileged actions on a member account in your organization. You can use the following steps to identify the actions taken by the management account or a delegated administrator during the `sts:AssumeRoot` session.

> **Note**
>
> The global endpoint is not supported for `sts:AssumeRoot`. CloudTrail records `ConsoleLogin` events in the Region specified for the endpoint.

To track actions performed by a privileged session in CloudTrail logs

1. Find the `AssumeRoot` event in your CloudTrail logs. This event is generated when your management account or the delegated administrator for IAM gets a set of short term credentials from `sts:AssumeRoot`.

   In the following example, the CloudTrail event for AssumeRoot is logged in the `eventName` field.

   ```
   {
       "eventVersion": "1.08",
       "userIdentity": {
           "type": "AssumedRole",
           "principalId": "AIDACKCEVSQ6C2EXAMPLE:JohnRole1",
   ```

```
        "arn": "arn:aws:sts::111111111111:assumed-role/John/JohnRole1",
        "accountId": "111111111111",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111111111111:role/John",
                "accountId": "111111111111",
                "userName": "Admin2"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-10-25T20:45:28Z",
                "mfaAuthenticated": "false"
            },
            "assumedRoot": "true"
        }
    },
    "eventTime": "2024-10-25T20:52:11Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRoot",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "requestParameters": {
        "targetPrincipal": "222222222222",
        "taskPolicyArn": {
            "arn": "arn:aws:iam::aws:policy/root-task/S3UnlockBucketPolicy"
        }
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
            "sessionToken": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
            "expiration": "Oct 25, 2024, 9:07:11 PM"
        }
    }
}
```


For steps to access your CloudTrail logs, see Getting and viewing your CloudTrail log files in the *AWS CloudTrail User Guide*.

2. In the CloudTrail event log, locate the `targetPrincipal` that specifies the member account actions were taken on, and the `accessKeyId` that is unique to the `AssumeRoot` session.

   In the following example, the `targetPrincipal` is 222222222222 and the `accessKeyId` is ASIAIOSFODNN7EXAMPLE.

```
"eventTime": "2024-10-25T20:52:11Z",
    "eventSource": "sts.amazonaws.com",
    "eventName": "AssumeRoot",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "requestParameters": {
        "targetPrincipal": "222222222222",
        "taskPolicyArn": {
            "arn": "arn:aws:iam::aws:policy/root-task/S3UnlockBucketPolicy"
        }
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
            "sessionToken": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
            "expiration": "Oct 25, 2024, 9:07:11 PM"
    }
}
```

3. In the CloudTrail logs for the target principal, search for the access key ID that corresponds to the `accessKeyId` value from the `AssumeRoot` event. Use the `eventName` field values to determine the privileged tasks performed during the `AssumeRoot` session. There may be multiple privileged tasks performed in a single session. The maximum session duration for `AssumeRoot` is 900 seconds (15 minutes).

   In the following example, the management account or delegated administrator deleted the resource-based policy for an Amazon S3 bucket.

```
{
    "eventVersion": "1.10",
    "userIdentity": {
        "type": "Root",
        "principalId": "222222222222",
        "arn": "arn:aws:iam::222222222222:root",
        "accountId": "222222222222",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
```

```
        "sessionContext": {
            "attributes": {
                "creationDate": "2024-10-25T20:52:11Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-10-25T20:53:47Z",
    "eventSource": "s3.amazonaws.com",
    "eventName": "DeleteBucketPolicy",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "requestParameters": {
        "bucketName": "resource-policy-John",
        "Host": "resource-policy-John.s3.amazonaws.com",
        "policy": ""
    },
    "responseElements": null,
    "requestID": "1234567890abcdef0",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "readOnly": false,
    "resources": [
        {
            "accountId": "222222222222",
            "type": "AWS::S3::Bucket",
            "ARN": "arn:aws:s3:::resource-policy-John"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "222222222222",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "resource-policy-John.s3.amazonaws.com"
    }
}
```

Compliance validation for AWS Identity and Access Management

Third-party auditors assess the security and compliance of AWS Identity and Access Management (IAM) as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, ISO, and others.

Our new AWS sign-up experience is not designed for regulated workloads. If you're using our new AWS sign-up experience, but you want to use AWS for regulated workloads, you can sign up for AWS (advanced) or activate advanced features for your AWS environment.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS Identity and Access Management

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions have multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) are self-sustaining, Region-based services that are available globally.

IAM is a critical AWS service. Every operation performed in AWS must be authenticated and authorized by IAM. IAM checks each request against the identities and policies stored in IAM to determine if the request is allowed or denied. IAM was designed with a separate *control plane* and *data plane* so that the service authenticates even during unexpected failures. IAM resources that

are used in authorizations, such as roles and policies, are stored in the control plane. IAM customers can change the configuration of these resources by using IAM operations such as `DeletePolicy` and `AttachRolePolicy`. Those configuration change requests go to the control plane. There is one IAM control plane for all commercial AWS Regions, which is located in the US East (N. Virginia) Region. The IAM system then propagates configuration changes to the IAM data planes in every enabled AWS Region. The IAM data plane is essentially a read-only replica of the IAM control plane configuration data. Each AWS Region has a completely independent instance of the IAM data plane, which performs authentication and authorization for requests from the same Region. In each Region, the IAM data plane is distributed across at least three Availability Zones, and has sufficient capacity to tolerate the loss of an Availability Zone without any customer impairment. Both the IAM control and data planes were built for *zero planned downtime*, with all software updates and scaling operations performed in a manner that is invisible to customers.

In Regions that are enabled by default, requests to the AWS STS global endpoint are automatically served in the same Region where the request originates. In opt-in Regions, requests to the AWS STS global endpoint are served by a single AWS Region, US East (N. Virginia). You can use a Regional AWS STS endpoint to reduce latency or provide additional redundancy for your applications. To learn more, see Manage AWS STS in an AWS Region.

Certain events can interrupt communication between AWS Regions over the network. However, even when you can't communicate with the global IAM endpoint, AWS STS can still authenticate IAM principals and IAM can authorize your requests. The specific details of an event that interrupts communication will determine your ability to access AWS services. In most situations, you can continue to use IAM credentials in your AWS environment. The following conditions might apply to an event that interrupts communication.

Access keys for IAM users

You can authenticate indefinitely in a Region with long-term access keys for IAM users. When you use the AWS Command Line Interface and APIs, you can provide AWS access keys so that AWS can verify your identity in programmatic requests.

> **Important**
>
> As a best practice, we recommend that your users sign in with temporary credentials instead of long-term access keys.

Temporary credentials

You can request new temporary credentials with the AWS STS Regional service endpoint for at least 24 hours. The following API operations generate temporary credentials.

- **AssumeRole**
- **AssumeRoleWithWebIdentity**
- **AssumeRoleWithSAML**
- **GetFederationToken**
- **GetSessionToken**

Principals and permissions

- You might not be able to add, modify, or remove principals or permissions in IAM.
- Your credentials might not reflect changes to your permissions that you recently applied in IAM. For more information, see Changes that I make are not always immediately visible.

AWS Management Console

- You might be able to use a Regional sign-in endpoint to sign in to the AWS Management Console as an IAM user. Regional sign-in endpoints have the following URL format.

  `https://{Account ID}.signin.aws.amazon.com/console?region={Region}`

  Example: https://111122223333.signin.aws.amazon.com/console?region=us-west-2

- You might not be able to complete Universal 2nd Factor (U2F) multi-factor authentication (MFA).

Best practices for IAM resilience

AWS has built resilience into AWS Regions and Availability Zones. When you observe the following IAM best practices in the systems that interact with your environment, you take advantage of that resilience.

1. Use an AWS STS Regional service endpoint instead of the default global endpoint.
2. Review the configuration of your environment for vital resources that routinely create or modify IAM resources, and prepare a fallback solution that uses existing IAM resources.

Infrastructure security in AWS Identity and Access Management

As a managed service, AWS Identity and Access Management is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access IAM through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

IAM can be accessed programmatically by using the IAM HTTPS API, which lets you issue HTTPS requests directly to the service. The Query API returns sensitive information, including security credentials. Therefore, you must use HTTPS with all API requests. When you use the HTTPS API, you must include code to digitally sign requests using your credentials.

You can call these API operations from any network location, but IAM does support resource-based access policies, which can include restrictions based on the source IP address. You can also use IAM policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given IAM resource from only the specific VPC within the AWS network.

Configuration and vulnerability analysis in AWS Identity and Access Management

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- Shared Responsibility Model
- Amazon Web Services: Overview of Security Processes (whitepaper)

The following resources also address configuration and vulnerability analysis in AWS Identity and Access Management (IAM):

- Compliance validation for AWS Identity and Access Management
- Security best practices and use cases in AWS Identity and Access Management

AWS managed policies for AWS Identity and Access Management, IAM Access Analyzer, and account access manager

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

IAMReadOnlyAccess

Use the `IAMReadOnlyAccess` managed policy to allow read only access to IAM resources. This policy grants permission to get and list all IAM resources. It allows viewing details and activity reports for users, groups, roles, policies, identity providers, and MFA devices. It does not include the

ability to create or delete resources or access to IAM Access Analyzer resources. View the policy for the full list of services and actions supported by this policy.

IAMUserChangePassword

Use the `IAMUserChangePassword` managed policy to allow IAM users to change their password.

You configure your IAM **Account settings** and the **Password policy** to allow IAM users to change their IAM account password. When you allow this action, IAM attaches the policy to each user.

To view the JSON policy, see IAMUserChangePassword in the AWS Managed Policy Reference Guide.

IAMAccessAnalyzerFullAccess

Use the `IAMAccessAnalyzerFullAccess` AWS managed policy to allow your administrators to access IAM Access Analyzer.

Permissions groupings

This policy is grouped into statements based on the set of permissions provided.

- **IAM Access Analyzer** – Allows full administrative permissions to all resources in IAM Access Analyzer.
- **Create service linked role** – Allows the administrator to create a service-linked role, which allows IAM Access Analyzer to analyze resources in other services on your behalf. This permission allows creating the service-linked role only for use by IAM Access Analyzer.
- **AWS Organizations** – Allows administrators to use IAM Access Analyzer for an organization in AWS Organizations. After enabling trusted access for IAM Access Analyzer in AWS Organizations, members of the management account can view findings across their organization.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "access-analyzer:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "access-analyzer.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeAccount",
        "organizations:DescribeOrganization",
        "organizations:DescribeOrganizationalUnit",
        "organizations:ListAccounts",
        "organizations:ListAccountsForParent",
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:ListChildren",
        "organizations:ListDelegatedAdministrators",
        "organizations:ListOrganizationalUnitsForParent",
        "organizations:ListParents",
        "organizations:ListRoots"
      ],
      "Resource": "*"
    }
  ]
}
```

IAMAccessAnalyzerReadOnlyAccess

Use the `IAMAccessAnalyzerReadOnlyAccess` AWS managed policy to allow read-only access to IAM Access Analyzer.

To also allow read-only access to IAM Access Analyzer for AWS Organizations, create a customer managed policy that allows the Describe and List actions from the IAMAccessAnalyzerFullAccess AWS managed policy.

Service-level permissions

This policy provides read-only access to IAM Access Analyzer. No other service permissions are included in this policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAMAccessAnalyzerReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "access-analyzer:CheckAccessNotGranted",
        "access-analyzer:CheckNoNewAccess",
        "access-analyzer:Get*",
        "access-analyzer:List*",
        "access-analyzer:ValidatePolicy"
      ],
      "Resource": "*"
    }
  ]
}
```

AccessAnalyzerServiceRolePolicy

You can't attach AccessAnalyzerServiceRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows IAM Access Analyzer to perform actions on your behalf. For more information, see Using service-linked roles for AWS Identity and Access Management and Access Analyzer.

Permissions groupings

This policy allows access to IAM Access Analyzer to analyze resource metadata from multiple AWS services.

- **Amazon DynamoDB** – Allows permissions to view DynamoDB streams and tables.
- **Amazon Elastic Compute Cloud** – Allows permissions to describe IP addresses, snapshots, and VPCs.

- **Amazon Elastic Container Registry** – Allows permissions to describe image repositories, retrieve account settings, and retrieve registry and repository policies.
- **Amazon Elastic File System** – Allows permissions to view the description of an Amazon EFS file system and view the resource-level policy for an Amazon EFS file system.
- **AWS Identity and Access Management** – Allows permissions to retrieve information about a specified role and list the IAM roles that have a specified path prefix. Allows permissions to retrieve information about users, IAM groups, login profiles, access keys, and service last accessed data.
- **AWS Key Management Service** – Allows permissions to view detailed information about an KMS key and its key policies and grants.
- **AWS Lambda** – Allows permissions to view information about Lambda aliases, functions, layers, and aliases.
- **AWS Organizations** – Allows permissions to AWS Organizations and allows the creation of an analyzer within the AWS organization as the zone of trust.
- **Amazon Relational Database Service** – Allows permissions to view detailed information about Amazon RDS DB snapshots and Amazon RDS DB cluster snapshots.
- **Amazon Simple Storage Service** – Allows permissions to view detailed information about Amazon S3 access points, buckets, Amazon S3 directory bucket access points, and directory buckets.
- **AWS Secrets Manager** – Allows permissions to view detailed information about secrets and resource policies attached to secrets.
- **Amazon Simple Notification Service** – Allows permissions to view detailed information about a topic.
- **Amazon Simple Queue Service** – Allows permissions to view detailed information about specified queues.

To view the JSON policy, see AccessAnalyzerServiceRolePolicy in the AWS Managed Policy Reference Guide.

IAMAuditRootUserCredentials

Use the `IAMAuditRootUserCredentials` AWS managed policy to scope down permissions when you perform a privileged task on an AWS Organizations member account to audit root user credential status of a member account. You can list or get individual root user credential information like:

- Whether there is a root user password
- If the root user has an access key and when it was last used
- If the root user has associated signing certificates
- The root user associated MFA devices
- List of the consolidated root user credential status

You can't attach `IAMAuditRootUserCredentials` to your IAM entities. This policy is attached to AssumeRoot to perform privileged tasks on a member account in your organization. For more information, see Centrally manage root access for member accounts.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyAllOtherActionsOnAnyResource",
         "NotAction": [
            "iam:ListAccessKeys",
            "iam:ListSigningCertificates",
            "iam:GetLoginProfile",
            "iam:ListMFADevices",
            "iam:GetAccountSummary",
            "iam:GetUser",
            "iam:GetAccessKeyLastUsed"
         ],
         "Effect": "Deny",
         "Resource": "*"
      },
      {
         "Sid": "DenyAuditingCredentialsOnNonRootUserResource",
         "Action": [
            "iam:ListAccessKeys",
            "iam:ListSigningCertificates",
            "iam:GetLoginProfile",
            "iam:ListMFADevices" ,
            "iam:GetUser",
            "iam:GetAccessKeyLastUsed"
         ],
         "Effect": "Deny",
```

```
            "NotResource": "arn:aws:iam::*:root"
        }
    ]
}
```

Permissions groupings

This policy is grouped into statements based on the set of permissions provided.

- **DenyAllOtherActionsOnAnyResource** – Denies access to credentials for all resources.
- **DenyAuditingCredentialsOnNonRootUserResource** – Denies access to credentials for all non-root user resources.

IAMCreateRootUserPassword

Use the `IAMCreateRootUserPassword` AWS managed policy to scope down permissions when you perform a privileged task on an AWS Organizations member account to allow password recovery for a member account with no root user credentials.

You can't attach `IAMCreateRootUserPassword` to your IAM entities. This policy is attached to AssumeRoot to perform privileged tasks on a member account in your organization. For more information, see Centrally manage root access for member accounts.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyAllOtherActionsOnAnyResource",
         "NotAction": [
            "iam:CreateLoginProfile",
            "iam:GetLoginProfile"
         ],
         "Effect": "Deny",
         "Resource": "*"
      },
      {
         "Sid": "DenyCreatingPasswordOnNonRootUserResource",
```

```
            "Action": [
               "iam:CreateLoginProfile",
               "iam:GetLoginProfile"
            ],
            "Effect": "Deny",
            "NotResource": "arn:aws:iam::*:root"
         }
      ]
}
```

Permissions groupings

This policy is grouped into statements based on the set of permissions provided.

- **DenyAllOtherActionsOnAnyResource** – Denies access to get or create a password for all resources.
- **DenyCreatingPasswordOnNonRootUserResource** – Denies access to get or create a password for all non-root user resources.

IAMDeleteRootUserCredentials

Use the `IAMDeleteRootUserCredentials` AWS managed policy to scope down permissions when you perform a privileged task on an AWS Organizations member account to remove root user credentials including password, access keys, signing certificates, and deactivating MFA. Additional permissions are required for this privileged action, so you can view last used credential information, verify last used information for the member account root user, and list permissions for all root user credentials to be deleted.

You can't attach `IAMDeleteRootUserCredentials` to your IAM entities. This policy is attached to AssumeRoot to perform privileged tasks on a member account in your organization. For more information, see Centrally manage root access for member accounts.

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
```

```
    "Sid": "DenyAllOtherActionsOnAnyResource",
    "Effect": "Deny",
    "NotAction": [
     "iam:DeleteAccessKey",
     "iam:DeleteSigningCertificate",
     "iam:DeleteLoginProfile",
     "iam:DeactivateMFADevice",
     "iam:ListAccessKeys",
     "iam:ListSigningCertificates",
     "iam:GetLoginProfile",
     "iam:ListMFADevices",
     "iam:GetUser",
     "iam:GetAccessKeyLastUsed"
    ],

    "Resource": "*"
   },
   {
    "Sid": "DenyDeletingRootUserCredentialsOnNonRootUserResource",
    "Effect": "Deny",
    "Action": [
     "iam:DeleteAccessKey",
     "iam:DeleteSigningCertificate",
     "iam:DeleteLoginProfile",
     "iam:DeactivateMFADevice",
     "iam:ListAccessKeys",
     "iam:ListSigningCertificates",
     "iam:GetLoginProfile",
     "iam:ListMFADevices",
     "iam:GetUser",
     "iam:GetAccessKeyLastUsed"
    ],

    "NotResource": "arn:aws:iam::*:root"
   }
  ]
}
```

Permissions groupings

This policy is grouped into statements based on the set of permissions provided.

- **DenyAllOtherActionsOnAnyResource** – Denies access to get or delete credentials for all resources.
- **DenyDeletingRootUserCredentialsOnNonRootUserResource** – Denies access to get or delete credentials for all non-root user resources.

S3UnlockBucketPolicy

Use the `S3UnlockBucketPolicy` AWS managed policy to scope down permissions when you perform a privileged task on an AWS Organizations member account to remove a misconfigured bucket policy that denies all principals from accessing an Amazon S3 bucket.

You can't attach `S3UnlockBucketPolicy` to your IAM entities. This policy is attached to AssumeRoot to perform privileged tasks on a member account in your organization. For more information, see Centrally manage root access for member accounts.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyAllOtherActionsOnAnyResource",
         "NotAction": [
            "s3:DeleteBucketPolicy",
            "s3:PutBucketPolicy",
            "s3:GetBucketPolicy",
            "s3:ListAllMyBuckets"
         ],
         "Effect": "Deny",
         "Resource": "*"
      },
      {
         "Sid": "DenyManagingBucketPolicyForNonRootCallers",
         "Action": [
            "s3:DeleteBucketPolicy",
            "s3:PutBucketPolicy",
            "s3:GetBucketPolicy",
            "s3:ListAllMyBuckets"
         ],
         "Effect": "Deny",
```

```
            "Resource": "*",
            "Condition" : {
               "ArnNotLike" : {
                  "aws:PrincipalArn" : "arn:aws:iam::*:root"
               }
            }
         }
      ]
}
```

Permissions groupings

This policy is grouped into statements based on the set of permissions provided.

- **DenyAllOtherActionsOnAnyResource** – Denies access to bucket policies for all resources.
- **DenyManagingBucketPolicyForNonRootCallers** – Denies access to bucket policies for all non-root user resources.

SQSUnlockQueuePolicy

Use the `SQSUnlockQueuePolicy` AWS managed policy to scope down permissions when you perform a privileged task on an AWS Organizations member account to delete an Amazon Simple Queue Service resource-based policy that denies all principals from accessing an Amazon SQS queue.

You can't attach `SQSUnlockQueuePolicy` to your IAM entities. This policy is attached to AssumeRoot to perform privileged tasks on a member account in your organization. For more information, see Centrally manage root access for member accounts.

Permissions groupings

This policy is grouped into statements based on the set of permissions provided.

- **DenyAllOtherActionsOnAnyResource** – Denies access to Amazon SQS actions for all resources.
- **DenyGettingQueueAttributesOnNonOwnQueue** – Denies access to Amazon SQS queue attributes for queues owned by another account.
- **DenyActionsForNonRootUser** – Denies access to Amazon SQS actions for all non-root user resources.

AccountAccessManagerServiceRolePolicy

You can't attach `AccountAccessManagerServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows account access manager to perform actions on your behalf. For more information, see the section called “Security”.

Service-level permissions

This policy allows account access manager to retrieve information about your Organizations accounts to maintain accurate account membership for access management. Account access manager uses these permissions to detect when accounts leave your organization, verify whether trusted service access is enabled, and confirm account status before managing entitlements.

- **AWS Organizations** – Allows permissions to list accounts in the organization, verify whether trusted service access is enabled, and describe account status.

To view the JSON policy, see AccountAccessManagerServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

IAM, IAM Access Analyzer and account access manager updates to AWS managed policies

View details about updates to IAM and AWS managed policies since the service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the IAM and IAM Access Analyzer Document history pages.

Change	Description	Date
AccountAccessManagerServiceRolePolicy – Added managed policy	Account access manager added a service-linked role policy that allows the service to retrieve information about your Organizations accounts.	July 31, 2026

Change	Description	Date
IAMUserChangePassword – Added permissions	IAM added permissions to `IAMUserChangePassword` to allow users specified within a path.	May 28, 2025
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added `iam:GetAccountAuthorizationDetails` to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	May 12, 2025
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for Amazon S3 directory bucket access points to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	March 31, 2025
IAMDeleteRootUserCredentials – Removed permissions	IAM removed the `iam:DeleteVirtualMFADevice` permission from the managed policy.	January 7, 2025
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve information about Amazon ECR account settings and registry policies to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	December 10, 2024

Change	Description	Date
IAMAuditRootUserCredentials – Added managed policy	IAM added managed policies for Centrally manage root access for member accounts to scope privileged tasks you can perform on AWS Organizations member accounts.	November 14, 2024
IAMCreateRootUserPassword – Added managed policy	IAM added managed policies for Centrally manage root access for member accounts to scope privileged tasks you can perform on AWS Organizations member accounts.	November 14, 2024
IAMDeleteRootUserCredentials – Added managed policy	IAM added managed policies for Centrally manage root access for member accounts to scope privileged tasks you can perform on AWS Organizations member accounts.	November 14, 2024
S3UnlockBucketPolicy – Added managed policy	IAM added managed policies for Centrally manage root access for member accounts to scope privileged tasks you can perform on AWS Organizations member accounts.	November 14, 2024

Change	Description	Date
SQSUnlockQueuePolicy – Added managed policy	IAM added managed policies for Centrally manage root access for member accounts to scope privileged tasks you can perform on AWS Organizations member accounts.	November 14, 2024
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve information about IAM user and role tags to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	October 29, 2024
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve information about IAM user and role policies to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	May 30, 2024
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve the current state of the block public access for Amazon EC2 snapshots to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	January 23, 2024

Change	Description	Date
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for DynamoDB streams and tables to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	January 11, 2024
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for Amazon S3 directory buckets to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	December 1, 2023
IAMAccessAnalyzerReadOnlyAccess – Added permissions	IAM Access Analyzer added permissions to allow you to check whether updates to your policies grant additional access. This permission is required by IAM Access Analyzer to perform policy checks on your policies.	November 26, 2023

Change	Description	Date
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added IAM actions to the service-level permissions of `AccessAnalyzerServiceRolePolicy` to support the following actions: • Listing entities for a policy • Generating service last accessed details • Listing access key information	November 26, 2023
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for the following resource types to the service-level permissions of `AccessAnalyzerServiceRolePolicy` : • Amazon EBS volume snapshots • Amazon ECR repositories • Amazon EFS file systems • Amazon RDS DB snapshots • Amazon RDS DB cluster snapshots • Amazon SNS topics	October 25, 2022

Change	Description	Date
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added the `lambda:GetFunctionUrlConfig` action to the service-level permissions of `AccessAnalyzerServiceRolePolicy` .	April 6, 2022
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added new Amazon S3 actions to analyze metadata associated with multi-region access points.	September 2, 2021
IAMAccessAnalyzerReadOnlyAccess – Added permissions	IAM Access Analyzer added a new action to grant `ValidatePolicy` permissions to allow you to use the policy checks for validation. This permission is required by IAM Access Analyzer to perform policy checks on your policies.	March 16, 2021
IAM Access Analyzer started tracking changes	IAM Access Analyzer started tracking changes for its AWS managed policies.	March 1, 2021

Security features outside IAM

You use IAM to control access to tasks that are performed using the AWS Management Console, the AWS Command Line Tools, or service API operations using the AWS SDKs. Some AWS products have other ways to secure their resources as well. The following list provides some examples, though it is not exhaustive.

Amazon EC2

In Amazon Elastic Compute Cloud you log into an instance with a key pair (for Linux instances) or using a user name and password (for Microsoft Windows instances).

For more information, see the following documentation:

- Getting Started with Amazon EC2 Linux Instances in the *Amazon EC2 User Guide*
- Getting Started with Amazon EC2 Windows Instances in the *Amazon EC2 User Guide*

Amazon RDS

In Amazon Relational Database Service you log into the database engine with a user name and password that are tied to that database.

For more information, see Getting Started with Amazon RDS in the *Amazon RDS User Guide*.

Amazon EC2 and Amazon RDS

In Amazon EC2 and Amazon RDS you use security groups to control traffic to an instance or database.

For more information, see the following documentation:

- Amazon EC2 Security Groups for Linux Instances in the *Amazon EC2 User Guide*
- Amazon EC2 Security Groups for Windows Instances in the *Amazon EC2 User Guide*
- Amazon RDS Security Groups in the *Amazon RDS User Guide*

WorkSpaces

In Amazon WorkSpaces, users sign in to a desktop with a user name and password.

For more information, see Getting Started with WorkSpaces in the *Amazon WorkSpaces Administration Guide*.

These access control methods are not part of IAM. IAM lets you control how these AWS products are administered—creating or terminating an Amazon EC2 instance, setting up new WorkSpaces desktops, and so on. That is, IAM helps you control the tasks that are performed by making requests to Amazon Web Services, and it helps you control access to the AWS Management Console. However, IAM does not help you manage security for tasks like signing in to an operating system (Amazon EC2), database (Amazon RDS), or desktop (Amazon WorkSpaces).

When you work with a specific AWS product, be sure to read the documentation to learn the security options for all the resources that belong to that product.

Using AWS Identity and Access Management Access Analyzer

AWS Identity and Access Management Access Analyzer provides the following capabilities:

- IAM Access Analyzer external access analyzers help identify resources in your organization and accounts that are shared with an external entity.
- IAM Access Analyzer internal access analyzers help identify internal access to selected resources in your organization and accounts.
- IAM Access Analyzer unused access analyzers help identify unused access in your organization and accounts.
- IAM Access Analyzer validates IAM policies against policy grammar and AWS best practices.
- IAM Access Analyzer custom policy checks help validate IAM policies against your specified security standards.
- IAM Access Analyzer generates IAM policies based on access activity in your AWS CloudTrail logs.

Identifying resources shared with an external entity

IAM Access Analyzer helps you identify the resources in your organization and accounts, such as Amazon S3 buckets or IAM roles, shared with an external entity. This lets you identify unintended access to your resources and data, which is a security risk. IAM Access Analyzer identifies resources shared with external principals by using logic-based reasoning to analyze the resource-based policies in your AWS environment. For each instance of a resource shared outside of your account, IAM Access Analyzer generates a finding. Findings include information about the access and the external principal granted to it. You can review findings to determine if the access is intended and safe or if the access is unintended and a security risk. In addition to helping you identify resources shared with an external entity, you can use IAM Access Analyzer findings to preview how your policy affects public and cross-account access to your resource before deploying resource permissions. The findings are organized in a visual summary dashboard. The dashboard highlights the split between public and cross-account access findings, and provides a breakdown of findings by resource type. To learn more about the dashboard, see View the IAM Access Analyzer findings dashboard.

> **Note**
>
> An external entity can be another AWS account, a root user, an IAM user or role, a federated user, an anonymous user, or another entity that you can use to create a filter. For more information, see AWS JSON Policy Elements: Principal.

For instructions on getting started, see Create an IAM Access Analyzer external access analyzer. After you create an analyzer, you can Manage an IAM Access Analyzer external access analyzer.

When you enable IAM Access Analyzer, you create an analyzer for your entire organization or your account. The organization or account you choose is known as the zone of trust for the analyzer. The analyzer monitors all of the supported resources within your zone of trust. Any access to resources by principals within your zone of trust is considered trusted. Once enabled, IAM Access Analyzer analyzes the policies applied to all of the supported resources in your zone of trust. After the first analysis, IAM Access Analyzer analyzes these policies periodically. If you add a new policy or change an existing policy, IAM Access Analyzer analyzes the new or updated policy within about 30 minutes.

When analyzing the policies, if IAM Access Analyzer identifies one that grants access to an external principal that isn't within your zone of trust, it generates a finding. Each finding includes details about the resource, the external entity with access to it, and the permissions granted so that you can take appropriate action. You can view the details included in the finding to determine whether the resource access is intentional or a potential risk that you should resolve. When you add a policy to a resource, or update an existing policy, IAM Access Analyzer analyzes the policy. IAM Access Analyzer also analyzes all resource-based policies periodically.

On rare occasions under certain conditions, IAM Access Analyzer does not receive notification of an added or updated policy, which can cause delays in generated findings. IAM Access Analyzer can take up to 6 hours to generate or resolve findings if you create or delete a multi-region access point associated with an Amazon S3 bucket, or update the policy for the multi-region access point. Also, if there is a delivery issue with AWS CloudTrail log delivery or resource control policy (RCP) restriction changes, the policy change does not trigger a rescan of the resource reported in the finding. When this happens, IAM Access Analyzer analyzes the new or updated policy during the next periodic scan, which is within 24 hours. If you want to confirm a change you make to a policy resolves an access issue reported in a finding, you can rescan the resource reported in a finding by using the **Rescan** link in the **Findings** details page, or by using the `StartResourceScan` operation of the IAM Access Analyzer API. To learn more, see Resolve IAM Access Analyzer findings.

Important

For external access, IAM Access Analyzer analyzes only policies applied to resources in the same AWS Region where it's enabled. To monitor all resources in your AWS environment, you must create an external access analyzer to enable IAM Access Analyzer in each Region where you're using supported AWS resources.
For unused access, findings for the analyzer do not change based on Region. Creating an unused access analyzer in each Region where you have resources is not required.

IAM Access Analyzer analyzes the following resource types for external access:

- Amazon Simple Storage Service buckets
- Amazon Simple Storage Service directory buckets
- AWS Identity and Access Management roles
- AWS Key Management Service keys
- AWS Lambda functions and layers
- Amazon Simple Queue Service queues
- AWS Secrets Manager secrets
- Amazon Simple Notification Service topics
- Amazon Elastic Block Store volume snapshots
- Amazon Relational Database Service DB snapshots
- Amazon Relational Database Service DB cluster snapshots
- Amazon Elastic Container Registry repositories
- Amazon Elastic File System file systems
- Amazon DynamoDB streams
- Amazon DynamoDB tables

Identifying internal access to business-critical resources

For your selected business-critical resources, IAM Access Analyzer helps you identify which principals within your organization or account have access to them. This analysis supports implementing the principle of least privilege by ensuring that your specified resources can only be accessed by the intended principals within your organization.

For instructions on getting started, see Create an IAM Access Analyzer internal access analyzer. After you create an analyzer, you can Manage an IAM Access Analyzer internal access analyzer.

Internal access analysis helps you:

- Determine which IAM users or roles within your account or organization can access your specified resources
- Understand access paths between principals and resources within your AWS environment
- Verify that your access controls are working as intended
- Review findings to determine if the access is intended and safe or if the access is unintended and presents a security risk
- Identify and remediate unintended access within your organization

IAM Access Analyzer analyzes the following resource types for internal access:

- Amazon Simple Storage Service buckets
- Amazon Simple Storage Service directory buckets
- Amazon Relational Database Service DB snapshots
- Amazon Relational Database Service DB cluster snapshots
- Amazon DynamoDB streams
- Amazon DynamoDB tables

Identifying unused access granted to IAM users and roles

IAM Access Analyzer helps you identify and review unused access in your AWS organization and accounts. IAM Access Analyzer continuously monitors all IAM roles and users in your AWS organization and accounts and generates findings for unused access. The findings highlight unused roles, unused access keys for IAM users, and unused passwords for IAM users. For active IAM roles and users, the findings provide visibility into unused services and actions.

For instructions on creating an unused access analyzer and managing its findings, see Manage an IAM Access Analyzer unused access analyzer.

IAM Access Analyzer reviews last accessed information for all roles in your AWS organization and accounts to help you identify unused access. IAM action last accessed information helps

you identify unused actions for roles in your AWS accounts. For more information, see Refine permissions in AWS using last accessed information.

The findings for external, internal, and unused access analyzers are organized into a visual summary dashboard. The dashboard highlights your AWS resources and AWS accounts that have the most findings and provides a breakdown of findings by type. For more information about the dashboard, see View the IAM Access Analyzer findings dashboard.

Validating policies against AWS best practices

You can validate your policies against IAM policy grammar and AWS best practices using the basic policy checks provided by IAM Access Analyzer policy validation. You can create or edit a policy using the AWS CLI, AWS API, or JSON policy editor in the IAM console. You can view policy validation check findings that include security warnings, errors, general warnings, and suggestions for your policy. These findings provide actionable recommendations that help you author policies that are functional and conform to AWS best practices. To learn more about validating policies using policy validation, see Validate policies with IAM Access Analyzer. For a list of the policy checks that IAM Access Analyzer performs, see Checks for validating policies.

Validating policies against your specified security standards

You can validate your policies against your specified security standards using the IAM Access Analyzer custom policy checks. You can create or edit a policy using the AWS CLI, AWS API, or JSON policy editor in the IAM console. Through the console, you can check whether your updated policy grants new access compared to the existing version. Through AWS CLI and AWS API, you can also check specific IAM actions that you consider critical are not allowed by a policy. These checks highlight a policy statement that grants new access. You can update the policy statement and re-run the checks until the policy conform to your security standard. To learn more about validating policies using custom policy checks, see Validate policies with IAM Access Analyzer custom policy checks.

Generating policies

IAM Access Analyzer analyzes your AWS CloudTrail logs to identify actions and services that have been used by an IAM entity (user or role) within your specified date range. It then generates an IAM policy that is based on that access activity. You can use the generated policy to refine an entity's

permissions by attaching it to an IAM user or role. To learn more about generating policies using IAM Access Analyzer, see IAM Access Analyzer policy generation.

Pricing for IAM Access Analyzer

IAM Access Analyzer charges for unused access analysis based on the number of IAM roles and users analyzed per analyzer per month.

- You will be charged for each unused access analyzer that you create.
- Creating unused access analyzers across multiple Regions will result in you being charged for each analyzer.
- Service-linked roles aren't analyzed for unused access activity and they aren't included in the total number of IAM roles analyzed.

IAM Access Analyzer charges for internal access analysis based on the number of resources monitored per internal access analyzer per month.

IAM Access Analyzer charges for custom policy checks based on the number of API requests made to IAM Access Analyzer to check for new access.

For a complete list of prices and charges for IAM Access Analyzer, see IAM Access Analyzer pricing.

To see your bill, go to the **Billing and Cost Management Dashboard** in the AWS Billing and Cost Management console. Your bill contains links to usage reports that provide details about your bill. To learn more about AWS account billing, see the AWS Billing User Guide

If you have questions concerning AWS billing, accounts, and events, contact Support.

IAM Access Analyzer findings

IAM Access Analyzer generates findings for external access, internal access, and unused access in your AWS account or organization.

For external access, IAM Access Analyzer generates a finding for each instance of a resource-based policy that grants access to a resource within your zone of trust to a principal that is not within your zone of trust. When you create an external access analyzer, you choose an organization or AWS account to analyze. Any principal in the organization or account that you choose for the

analyzer is considered trusted. Because principals in the same organization or account are trusted, the resources and principals within the organization or account comprise the zone of trust for the analyzer. Any sharing that is within the zone of trust is considered safe, so IAM Access Analyzer does not generate a finding. For example, if you select an organization as the zone of trust for an analyzer, all resources and principals in the organization are within the zone of trust. If you grant permissions to an Amazon S3 bucket in one of your organization member accounts to a principal in another organization member account, IAM Access Analyzer does not generate a finding. But if you grant permission to a principal in an account that is not a member of the organization, IAM Access Analyzer generates a finding.

For internal access, IAM Access Analyzer generates findings when there is a possible access path between an IAM role or user within your organization and your specified resources. Similar to external access analysis, the scope you choose (organization or account) determines what is considered internal. If you select an organization as the scope, IAM Access Analyzer will generate findings for access paths between principals and resources within your organization. If you select an account, findings will be generated for access paths within that specific account. IAM Access Analyzer uses automated reasoning to evaluate all IAM policies to monitor who has access to your resources.

The combination of external and internal access findings with the same zone of trust provides a comprehensive analysis of all possible access to a particular resource, both from within and outside your defined trust boundary.

For unused access, IAM Access Analyzer generates findings for unused access granted in your AWS organization and accounts. When you create an unused access analyzer, IAM Access Analyzer continuously monitors all IAM roles and users in your AWS organization and accounts and generates findings for unused access. IAM Access Analyzer generates the following types of findings for unused access:

- **Unused roles** – Roles with no access activity within the specified usage window.
- **Unused IAM user access keys and passwords** – Credentials belonging to IAM users that have not been used to access your AWS account in the specified usage window.
- **Unused permissions** – Service-level and action-level permissions that weren't used by a role within the specified usage window. IAM Access Analyzer uses identity-based policies attached to roles to determine the services and actions that those roles can access. IAM Access Analyzer supports review of unused permissions for all service-level permissions. For a complete list of action-level permissions that are supported for unused access findings, see IAM action last accessed information services and actions.

> **Note**
>
> IAM Access Analyzer offers external access findings for free. There are charges for unused access findings based on the number of IAM roles and users analyzed per analyzer per month. There are also charges for internal access findings based on the number of AWS resources monitored per analyzer per month. For more details about pricing, see IAM Access Analyzer pricing.

Topics

- Understand how IAM Access Analyzer findings work
- Getting started with AWS Identity and Access Management Access Analyzer
- View the IAM Access Analyzer findings dashboard
- Review IAM Access Analyzer findings
- Filter IAM Access Analyzer findings
- Archive IAM Access Analyzer findings
- Resolve IAM Access Analyzer findings
- IAM Access Analyzer error findings
- IAM Access Analyzer supported resource types for external and internal access
- Delegated administrator for IAM Access Analyzer
- Archive rules
- Monitoring AWS Identity and Access Management Access Analyzer with Amazon EventBridge
- Integrate IAM Access Analyzer with AWS Security Hub CSPM
- Logging IAM Access Analyzer API calls with AWS CloudTrail
- IAM Access Analyzer filter keys
- Using service-linked roles for AWS Identity and Access Management Access Analyzer

Understand how IAM Access Analyzer findings work

This topic describes the concepts and terms that are used in IAM Access Analyzer to help you become familiar with how IAM Access Analyzer monitors access to your AWS resources.

External access findings

External access findings are generated only once for each instance of a resource that is shared outside of your zone of trust. Each time a resource-based policy is modified, IAM Access Analyzer analyzes the policy. If the updated policy shares a resource that is already identified in a finding, but with different permissions or conditions, a new finding is generated for that instance of the resource sharing. Changes to a resource control policy that impact the **Resource control policy (RCP) restriction** also generate a new finding. IAM Access Analyzer also evaluates access control configurations established by declarative policies. If the access in the first finding is removed, that finding is updated to a status of **Resolved**.

The status of all findings remains **Active** until you archive them or remove the access that generated the finding. When you remove the access, the finding status is updated to **Resolved**.

> **Note**
>
> It may take up to 30 minutes after a policy is modified for IAM Access Analyzer to analyze the resource and then update the external access finding. Changes to a resource control policy (RCP) do not trigger a rescan of the resource reported in the finding. IAM Access Analyzer analyzes the new or updated policy during the next periodic scan, which is within 24 hours.

How IAM Access Analyzer generates findings for external access

AWS Identity and Access Management Access Analyzer uses a technology called Zelkova to analyze IAM policies and identify external access to resources.

Zelkova translates IAM policies into equivalent logical statements and runs them through a suite of general-purpose and specialized logical solvers (satisfiability modulo theories). IAM Access Analyzer applies Zelkova repeatedly to a policy, using increasingly specific queries to characterize the types of access the policy allows based on its content. For more information about satisfiability modulo theories, see Satisfiability Modulo Theories.

For external access analyzers, IAM Access Analyzer does not examine access logs to determine whether an external entity has actually accessed a resource within your zone of trust. Instead, it generates a finding when a resource-based policy allows access to a resource, regardless of whether the resource was accessed by the external entity.

Additionally, IAM Access Analyzer does not consider the state of any external accounts when making its determinations. If it indicates that account 111122223333 can access your Amazon S3 bucket, it doesn't have any information about the users, roles, service control policies (SCP), or other relevant configurations in that account. This is for customer privacy, as IAM Access Analyzer doesn't know who owns the other account. This is also for security, as it's important to know about potential external access even if there are currently no active principals that can use it.

IAM Access Analyzer only considers certain IAM condition keys that external users can't directly influence or that are otherwise impactful to authorization. For examples of condition keys IAM Access Analyzer considers, see IAM Access Analyzer filter keys.

IAM Access Analyzer doesn't currently report findings from AWS service principals or internal service accounts. In rare cases where it can't fully determine whether a policy statement grants access to an external entity, it errs on the side of declaring a false positive finding. This is because IAM Access Analyzer is designed to provide a comprehensive view of the resource sharing in your account and to minimize false negatives.

Internal access findings

To use internal access analysis, you must first configure the analyzer by selecting the specific resources you want to monitor. Once configured, internal access findings are generated when a principal (IAM user or role) within your organization or account has access to your selected resources. A new finding is generated the next time the analyzer scans the specified resources and identifies a principal that has access to the resources. If an updated policy allows a principal that is already identified in a finding, but with different permissions or conditions, a new finding is generated for that instance of the principal and resource. This updated policy could be a resource-based policy, identity-based policy, service control policy (SCP), or resource control policy (RCP). IAM Access Analyzer also evaluates access control configurations established by declarative policies.

> **Note**
>
> Internal access findings are only available using the ListFindingsV2 API action.

How IAM Access Analyzer generates findings for internal access

To analyze internal access, you must create a separate analyzer for internal access findings for your resources, even if you've already created an analyzer to generate external access findings or unused access findings.

After creating the internal access analyzer, IAM Access Analyzer evaluates all resource-based policies, identity-based policies, service control policies (SCPs), resource control policies (RCPs), and permissions boundaries within your specified account or organization.

By creating an analyzer dedicated to internal access to your selected resources, you can identify:

- When a principal in your organization or account can access your selected resources
- The total effective permissions allowed for a principal based on the intersection of all applicable policies
- Complex access paths where a principal gains access based on the combination of identity policies and resource policies

> **Note**
>
> IAM Access Analyzer cannot generate internal access findings for organizations that contain more than 70,000 principals (IAM users and roles combined).

Unused access findings

Unused access findings are generated for IAM entities (principals) within the selected account or organization based on the number of days specified while creating the analyzer. A new finding is generated the next time the analyzer scans the entities if one of the following conditions is met:

- A role is inactive for the specified number of days.
- An unused permission, unused user password, or unused user access key surpasses the specified number of days.

> **Note**
>
> Unused access findings are only available using the ListFindingsV2 API action.

How IAM Access Analyzer generates findings for unused access

To analyze unused access, you must create a separate analyzer for unused access findings for your roles, even if you've already created an analyzer to generate external or internal access findings for your resources.

After creating the unused access analyzer, IAM Access Analyzer reviews access activity to identify unused access. IAM Access Analyzer examines the last accessed information for all IAM users, IAM roles including service roles, user access keys, and user passwords across your AWS organization and accounts. This helps you identify unused access.

> **Note**
>
> A service-linked role is a special type of service role that is linked to an AWS service and owned by the service. Service-linked roles are not analyzed by unused access analyzers.

For active IAM roles and users, IAM Access Analyzer uses last accessed information for IAM services and actions to identify unused permissions. This allows you to scale your review process at the AWS organization and account level. You can also use the action last accessed information for deeper investigation of individual roles. This provides more granular insights into which specific permissions are not being utilized.

By creating an analyzer dedicated to unused access, you can comprehensively review and identify unused access across your AWS environment, complementing the findings generated by your existing external access analyzer.

Getting started with AWS Identity and Access Management Access Analyzer

Use the information in this topic to learn about the requirements necessary to use and manage AWS Identity and Access Management Access Analyzer.

If you are using IAM Access Analyzer within an AWS Organizations organization, you can Add a delegated administrator for IAM Access Analyzer. For information about service limits, see IAM Access Analyzer quotas.

Permissions required to use IAM Access Analyzer

To successfully configure and use IAM Access Analyzer, the account you use must be granted the required permissions.

AWS managed policies for IAM Access Analyzer

AWS Identity and Access Management Access Analyzer provides AWS managed policies to help you get started quickly.

- IAMAccessAnalyzerFullAccess - Allows full access to IAM Access Analyzer for administrators. This policy also allows creating the service-linked roles that are required to allow IAM Access Analyzer to analyze resources in your account or AWS organization.
- IAMAccessAnalyzerReadOnlyAccess - Allows read-only access to IAM Access Analyzer. You must add additional policies to your IAM identities (users, groups of users, or roles) to allow them to view their findings.

Resources defined by IAM Access Analyzer

To view the resources defined by IAM Access Analyzer, see Resource types defined by IAM Access Analyzer in the *Service Authorization Reference*.

Required IAM Access Analyzer service permissions

IAM Access Analyzer uses a service-linked role (SLR) named `AWSServiceRoleForAccessAnalyzer`. This SLR grants the service read-only access to analyze AWS resources with resource-based policies and analyze unused access on your behalf. The service creates the role in your account in the following scenarios:

- You create an external access analyzer with your account as the zone of trust.
- You create an unused access analyzer with your account as the selected account.
- You create an internal access analyzer with your account as the zone of trust.

For more information, see Using service-linked roles for AWS Identity and Access Management Access Analyzer.

> **Note**
>
> IAM Access Analyzer is Regional. For external and internal access, you must enable IAM Access Analyzer in each Region independently.
> For unused access, findings for the analyzer do not change based on Region. Creating an analyzer in each Region where you have resources is not required.

In some cases, after you create an analyzer in IAM Access Analyzer, the **Findings** page or dashboard loads with no findings or summary. This might be due to a delay in the console for populating your findings. You might need to manually refresh the browser or check back later to view your findings or summary. If you still don't see any findings for an external access analyzer, it's because you have no supported resources in your account that can be accessed by an external entity. If a policy that grants access to an external entity is applied to a resource, IAM Access Analyzer generates a finding.

> **Note**
>
> For external access analyzers, it may take up to 30 minutes after a policy is modified for IAM Access Analyzer to analyze the resource and then either generate a new finding or update an existing finding for the access to the resource.
> When you create an internal access analyzer, it might take several minutes or hours before findings are available. After the initial scan, IAM Access Analyzer automatically rescans all policies every 24 hours.
> For all types of access analyzers, updates for findings might not be reflected in the dashboard immediately.

Required IAM Access Analyzer permissions to view the findings dashboard

To view the IAM Access Analyzer findings dashboard, the account you use must be granted access to perform the following required actions:

- `GetAnalyzer`
- `ListAnalyzers`
- `GetFindingsStatistics`

To view all of the actions defined by IAM Access Analyzer, see Actions defined by IAM Access Analyzer in the *Service Authorization Reference*.

IAM Access Analyzer status

To view the status of your analyzers, choose **Analyzers**. Analyzers created for an organization or account can have the following status:

Status	Description
Active	For external and internal access analyzers, the analyzer is actively monitoring resources within its zone of trust. The analyzer actively generates new findings and updates existing findings. For unused access analyzers, the analyzer is actively monitoring unused access within the selected organization or AWS account in the specified tracking period. The analyzer actively generates new findings and updates existing findings.
Creating	The creation of the analyzer is still in progress. The analyzer becomes active once creation is complete.
Disabled	The analyzer is disabled due to an action taken by the AWS Organizations administrator. For example, removing the analyzer's account as the delegated administrator for IAM Access Analyzer. When the analyzer is in a disabled state, it does not generate new findings or update existing findings.
Failed	The creation of the analyzer failed due to a configuration issue. The analyzer won't

Status	Description
	generate any findings. Delete the analyzer and create a new analyzer.

Create an IAM Access Analyzer external access analyzer

To enable an external access analyzer in a Region, you must create an analyzer in that Region. You must create an external access analyzer in each Region in which you want to monitor access to your resources.

> **Note**
>
> After you create or update an analyzer, it can take time for findings to be available.

Create an external access analyzer with the AWS account as the zone of trust

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. Choose **Create analyzer**.
4. In the **Analysis** section, choose **Resource analysis - External access**.
5. In the **Analyzer details** section, confirm that the Region displayed is the Region where you want to enable IAM Access Analyzer.
6. Enter a name for the analyzer.
7. Choose **Current account** as the zone of trust for the analyzer.

   > **Note**
   >
   > If your account is not the AWS Organizations management account or delegated administrator account, you can create only one analyzer with your account as the zone of trust.

8. Optional. Add any tags that you want to apply to the analyzer.
9. Choose **Create analyzer**.

When you create an external access analyzer to enable IAM Access Analyzer, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in your account.

Create an external access analyzer with the organization as the zone of trust

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. Choose **Create analyzer**.
4. In the **Analysis** section, choose **Resource analysis - External access**.
5. In the **Analyzer details** section, confirm that the Region displayed is the Region where you want to enable IAM Access Analyzer.
6. Enter a name for the analyzer.
7. Choose **Current organization** as the zone of trust for the analyzer.
8. Optional. Add any tags that you want to apply to the analyzer.
9. Choose **Submit**.

When you create an external access analyzer with the organization as the zone of trust, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in each account of your organization.

Manage an IAM Access Analyzer external access analyzer

To enable an external access analyzer in a Region, you must create an analyzer in that Region. You must create an external access analyzer in each Region in which you want to monitor access to your resources.

> **Note**
>
> After you create or update an analyzer, it can take time for findings to be available.

Update an external access analyzer

Use the following procedure to update an external access analyzer.

1. Open the IAM console at https://console.aws.amazon.com/iam/.

2. Under **Access analyzer**, choose **Analyzer settings**.
3. In the **Analyzers** section, choose the name of the external access analyzer to manage.
4. On the **Archive rules** tab, you can create, edit, or delete archive rules for the analyzer. For more information, see Archive rules.
5. On the **Tags** tab, you can manage and create tags for the analyzer. For more information, see Tags for AWS Identity and Access Management resources.

Delete an external access analyzer

Use the following procedure to delete an external access analyzer. When you delete an analyzer, the resources are no longer monitored and no new findings are generated. All findings that were generated by the analyzer are deleted.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. In the **Analyzers** section, choose the name of the external access analyzer to delete.
4. Choose **Delete analyzer**.
5. Enter **delete** and choose **Delete** to confirm deleting the analyzer.

Create an IAM Access Analyzer internal access analyzer

To enable an internal access analyzer in a Region, you must create an analyzer in that Region. You must create an internal access analyzer in each Region in which you want to monitor access to your resources.

IAM Access Analyzer charges for internal access analysis based on the number of resources monitored per analyzer per month. For more details about pricing, see IAM Access Analyzer pricing.

> **Note**
>
> After you create or update an analyzer, it can take time for findings to be available.
> IAM Access Analyzer cannot generate internal access findings for organizations that contain more than 70,000 principals (IAM users and roles combined).
> You can only create one organization-level internal access analyzer in an AWS organization.

Create an internal access analyzer with the AWS account as the zone of trust

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. Choose **Create analyzer**.
4. In the **Analysis** section, choose **Resource analysis - Internal access**.
5. In the **Analyzer details** section, confirm that the Region displayed is the Region where you want to enable IAM Access Analyzer.
6. Enter a name for the analyzer.
7. Choose **Current account** as the zone of trust for the analyzer.

> **Note**
>
> If your account is not the AWS Organizations management account or delegated administrator account, you can create only one analyzer with your account as the zone of trust.

8. In the **Resources to analyze** section, add resources for the analyzer to monitor.

   - To add resources by account, choose **Add > Add resources from selected accounts**.
     1. Choose **All supported resource types** or choose **Define specific resource types** and select the resource types from the **Resource type** list.

        Internal access analyzers support the following resource types:

        - Amazon Simple Storage Service buckets
        - Amazon Simple Storage Service directory buckets
        - Amazon Relational Database Service DB snapshots
        - Amazon Relational Database Service DB cluster snapshots
        - Amazon DynamoDB streams
        - Amazon DynamoDB tables

     2. Choose **Add resources**.
   - To add resources by Amazon Resource Name (ARN), choose **Add resources > Add resources by pasting in resource ARN**.

> **Note**
>
> ARNs must be exact matches – wildcards are not supported. For Amazon S3, only bucket ARNs are supported. Amazon S3 object ARNs and prefixes are not supported.

1. For each resource ARN, enter the account owner ID and the resource ARN separated by a comma. Enter one account owner ID and resource ARN per line.
2. Choose **Add resources**.

- To add resources by a CSV file, choose **Add resources > Add resources by uploading a CSV**.

  You can use AWS Resource Explorer to search for resources in your accounts and export a CSV file. Then you can upload the CSV file to configure the resources for the analyzer to monitor.

  1. Choose **Choose file** and select the CSV file from your computer.
  2. Choose **Add resources**.

9. Optional. Add any tags that you want to apply to the analyzer.
10. Choose **Create analyzer**.

When you create an internal access analyzer to enable IAM Access Analyzer, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in your account.

Create an internal access analyzer with the organization as the zone of trust

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. Choose **Create analyzer**.
4. In the **Analysis** section, choose **Resource analysis - Internal access**.
5. In the **Analyzer details** section, confirm that the Region displayed is the Region where you want to enable IAM Access Analyzer.
6. Enter a name for the analyzer.
7. Choose **Entire organization** as the zone of trust for the analyzer.
8. In the **Resources to analyze** section, add resources for the analyzer to monitor.

- To add resources for the account, choose **Add resources > Add resources from selected accounts**.

  1. Choose **All supported resource types** or choose **Define specific resource types** and select the resource types from the **Resource type** list.

     Internal access analyzers support the following resource types:

     - Amazon Simple Storage Service buckets
     - Amazon Simple Storage Service directory buckets
     - Amazon Relational Database Service DB snapshots
     - Amazon Relational Database Service DB cluster snapshots
     - Amazon DynamoDB streams
     - Amazon DynamoDB tables

  2. To select accounts from your organization, choose **Select from organization**. In the **Select accounts** section, choose **Hierarchy** to select accounts by organizational structure or **List** to select accounts from a list of all accounts in your organization.

     To manually enter accounts from your organization, choose **Enter AWS account ID**. Enter one or more AWS account IDs separated by commas in the **AWS account ID** field.

  3. Choose **Add resources**.

- To add resources by Amazon Resource Name (ARN), choose **Add resources > Add resources by pasting in resource ARN**.

  > **Note**
  >
  > ARNs must be exact matches – wildcards are not supported. For Amazon S3, only bucket ARNs are supported. Amazon S3 object ARNs and prefixes are not supported.

  1. For each resource ARN, enter the account owner ID and the resource ARN separated by a comma. Enter one account owner ID and resource ARN per line.
  2. Choose **Add resources**.

- To add resources by a CSV file, choose **Add resources > Add resources by uploading a CSV**.

You can use AWS Resource Explorer to search for resources in your accounts and export a CSV file. Then you can upload the CSV file to configure the resources for the analyzer to monitor.

1. Choose **Choose file** and select the CSV file from your computer.
2. Choose **Add resources**.

9. Optional. Add any tags that you want to apply to the analyzer.
10. Choose **Submit**.

When you create an internal access analyzer with the organization as the zone of trust, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in each account of your organization.

Manage an IAM Access Analyzer internal access analyzer

To enable an internal access analyzer in a Region, you must create an analyzer in that Region. You must create an internal access analyzer in each Region in which you want to monitor access to your resources.

> **Note**
>
> After you create or update an analyzer, it can take time for findings to be available.

Update an internal access analyzer

Use the following procedure to update an internal access analyzer.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. In the **Analyzers** section, choose the name of the internal access analyzer to manage.
4. On the **Archive rules** tab, you can create, edit, or delete archive rules for the analyzer. For more information, see Archive rules.
5. On the **Tags** tab, you can manage and create tags for the analyzer. For more information, see Tags for AWS Identity and Access Management resources.
6. On the **Resources** tab, choose **Edit** in the **Resources to analyze** section.

a. To add resources by account, choose **Add resources > Add resources from selected accounts**.

   1. Choose **All supported resource types** or choose **Define specific resource types** and select the resource types from the **Resource type** list.

      Internal access analyzers support the following resource types:

      - Amazon Simple Storage Service buckets
      - Amazon Simple Storage Service directory buckets
      - Amazon Relational Database Service DB snapshots
      - Amazon Relational Database Service DB cluster snapshots
      - Amazon DynamoDB streams
      - Amazon DynamoDB tables

   2. Choose **Add resources**.

b. To add resources by Amazon Resource Name (ARN), choose **Add resources > Add resources by pasting in resource ARN**.

   > **Note**
   >
   > ARNs must be exact matches – wildcards are not supported. For Amazon S3, only bucket ARNs are supported. Amazon S3 object ARNs and prefixes are not supported.

   1. For each resource ARN, enter the account owner ID and the resource ARN separated by a comma. Enter one account owner ID and resource ARN per line.
   2. Choose **Add resources**.

c. To add resources by a CSV file, choose **Add resources > Add resources by uploading a CSV**.

   You can use AWS Resource Explorer to search for resources in your accounts and export a CSV file. Then you can upload the CSV file to configure the resources for the analyzer to monitor.

   1. Choose **Choose file** and select the CSV file from your computer.

2. Choose **Add resources**.

d. To remove resources from the analyzer, select the check box next to the resources to remove and choose **Remove**.

e. Choose **Save changes**.

> **Note**
>
> Any updates to the analyzer will be evaluated at the next automatic rescan within 24 hours.

Delete an internal access analyzer

Use the following procedure to delete an internal access analyzer. When you delete an analyzer, the resources are no longer monitored and no new findings are generated. All findings that were generated by the analyzer are deleted.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. In the **Analyzers** section, choose the name of the internal access analyzer to delete.
4. Choose **Delete analyzer**.
5. Enter **delete** and choose **Delete** to confirm deleting the analyzer.

Create an IAM Access Analyzer unused access analyzer

Create an unused access analyzer for the current account

Use the following procedure to create an unused access analyzer for a single AWS account. For unused access, findings for the analyzer do not change based on Region. Creating an analyzer in each Region where you have resources is not required.

IAM Access Analyzer charges for unused access analysis based on the number of IAM roles and users analyzed per month per analyzer. For more details about pricing, see IAM Access Analyzer pricing.

> **Note**
>
> After you create or update an analyzer, it can take time for findings to be available.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. Choose **Create analyzer**.
4. In the **Analysis** section, choose **Principal analysis - Unused access**.
5. Enter a name for the analyzer.
6. For **Tracking period**, enter the number of days for analysis. The analyzer will only evaluate permissions for IAM entities within the selected account that have existed for the entire tracking period. For example, if you set a tracking period of 90 days, only permissions that are at least 90 days old will be analyzed, and findings will be generated if they show no usage during this period. You can enter a value between 1 and 365 days.
7. In the **Analyzer details** section, confirm that the Region displayed is the Region where you want to enable IAM Access Analyzer.
8. For **Selected accounts**, choose **Current account**.

   > **Note**
   >
   > If your account is not the AWS Organizations management account or delegated administrator account, you can create only one analyzer with your account as the selected account.

9. Optional. In the **Exclude IAM users and roles with tags** section, you can specify key-value pairs for IAM users and roles to exclude from unused access analysis. Findings will not be generated for excluded IAM users and roles that match the key-value pairs. For the **Tag key**, enter a value that is 1 to 128 characters in length and not prefixed with `aws:`. For the **Value**, you can enter a value that is 0 to 256 characters in length. If you don't enter a **Value**, the rule is applied to all principals with the specified **Tag key**. Choose **Add new exclusion** to add additional key-value pairs to exclude.
10. Optional. Add any tags that you want to apply to the analyzer.
11. Choose **Create analyzer**.

When you create an unused access analyzer to enable IAM Access Analyzer, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in your account.

Create an unused access analyzer with the current organization

Use the following procedure to create an unused access analyzer for an organization to centrally review all AWS accounts in an organization. For unused access analysis, findings for the analyzer do not change based on Region. Creating an analyzer in each Region where you have resources is not required.

IAM Access Analyzer charges for unused access analysis based on the number of IAM roles and users analyzed per month per analyzer. For more details about pricing, see IAM Access Analyzer pricing.

> **Note**
>
> If a member account is removed from the organization, the unused access analyzer will stop generating new findings and updating existing findings for that account after 24 hours. Findings associated with the member account that is removed from the organization will be removed permanently after 90 days.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. Choose **Create analyzer**.
4. In the **Analysis** section, choose **Principal analysis - Unused access**.
5. Enter a name for the analyzer.
6. For **Tracking period**, enter the number of days for analysis. The analyzer will only evaluate permissions for IAM entities within the accounts of the selected organization that have existed for the entire tracking period. For example, if you set a tracking period of 90 days, only permissions that are at least 90 days old will be analyzed, and findings will be generated if they show no usage during this period. You can enter a value between 1 and 365 days.
7. In the **Analyzer details** section, confirm that the Region displayed is the Region where you want to enable IAM Access Analyzer.
8. For **Selected accounts**, choose **Current organization**.

9. Optional. In the **Exclude AWS accounts from analysis** section, you can choose AWS accounts in your organization to exclude from unused access analysis. Findings will not be generated for excluded accounts.

   a. To specify individual account IDs to exclude, choose **Specify AWS account ID** and enter the account IDs separated by commas in the **AWS account ID** field. Choose **Exclude**. The accounts are then listed in the **AWS accounts to exclude** table.

   b. To choose from a list of accounts in your organization to exclude, choose **Choose from organization**.

      i. You can search for accounts by name, email, and account ID in the **Exclude accounts from organization** field.

      ii. Choose **Hierarchy** to view your accounts by organizational unit or choose **List** to view a list of all individual accounts in your organization.

      iii. Choose **Exclude all current accounts** to exclude all accounts in an organizational unit or choose **Exclude** to exclude individual accounts.

   The accounts are then listed in the **AWS accounts to exclude** table.

   > **Note**
   >
   > Excluded accounts cannot include the organization analyzer owner account. When new accounts are added to your organization, they are not excluded from analysis, even if you previously excluded all current accounts within an organizational unit. For more information on excluding accounts after creating an unused access analyzer, see Manage an IAM Access Analyzer unused access analyzer.

10. Optional. In the **Exclude IAM users and roles with tags** section, you can specify key-value pairs for IAM users and roles to exclude from unused access analysis. Findings will not be generated for excluded IAM users and roles that match the key-value pairs. For the **Tag key**, enter a value that is 1 to 128 characters in length and not prefixed with `aws:`. For the **Value**, you can enter a value that is 0 to 256 characters in length. If you don't enter a **Value**, the rule is applied to all principals with the specified **Tag key**. Choose **Add new exclusion** to add additional key-value pairs to exclude.
11. Optional. Add any tags that you want to apply to the analyzer.
12. Choose **Create analyzer**.

When you create an unused access analyzer to enable IAM Access Analyzer, a service-linked role named `AWSServiceRoleForAccessAnalyzer` is created in your account.

Manage an IAM Access Analyzer unused access analyzer

Use the information in this topic to learn about how to update or delete an existing unused access analyzer.

> **Note**
>
> After you create or update an analyzer, it can take time for findings to be available.

Update an unused access analyzer

Use the following procedure to update an unused access analyzer.

IAM Access Analyzer charges for unused access analysis based on the number of IAM roles and users analyzed per month per analyzer. For more details about pricing, see IAM Access Analyzer pricing.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Analyzer settings**.
3. In the **Analyzers** section, choose the name of the unused access analyzer to manage.
4. On the **Exclusion** tab, if the analyzer was created for an organization as the scope of analysis, choose **Manage** in the **Excluded AWS accounts** section.
    a. To specify individual account IDs to exclude, choose **Specify AWS account ID** and enter the account IDs separated by commas in the **AWS account ID** field. Choose **Exclude**. The accounts are then listed in the **AWS accounts to exclude** table.
    b. To choose from a list of accounts in your organization to exclude, choose **Choose from organization**.
        i. You can search for accounts by name, email, and account ID in the **Exclude accounts from organization** field.
        ii. Choose **Hierarchy** to view your accounts by organizational unit or choose **List** to view a list of all individual accounts in your organization.
        iii. Choose **Exclude all current accounts** to exclude all accounts in an organizational unit or choose **Exclude** to exclude individual accounts.

The accounts are then listed in the **AWS accounts to exclude** table.

c. To remove accounts to exclude, choose **Remove** next to the account in the **AWS accounts to exclude** table.

d. Choose **Save changes**.

> **Note**
>
> - Excluded accounts cannot include the organization analyzer owner account.
> - When new accounts are added to your organization, they are not excluded from analysis, even if you previously excluded all current accounts within an organizational unit.
> - After you update the exclusions for an analyzer, it can take up to two days for the list of excluded accounts to be updated.

5. On the **Exclusion** tab, choose **Manage** in the **Excluded IAM users and roles with tags** section.

   a. You can specify key-value pairs for IAM users and roles to exclude from unused access analysis. For the **Tag key**, enter a value that is 1 to 128 characters in length and not prefixed with `aws:`. For the **Value**, you can enter a value that is 0 to 256 characters in length. If you don't enter a **Value**, the rule is applied to all principals with the specified **Tag key**.

   b. Choose **Add new exclusion** to add additional key-value pairs to exclude.

   c. To remove key-value pairs to exclude, choose **Remove** next to the key-value pair.

   d. Choose **Save changes**.

6. On the **Archive rules** tab, you can create, edit, or delete archive rules for the analyzer. For more information, see Archive rules.

7. On the **Tags** tab, you can manage and create tags for the analyzer. For more information, see Tags for AWS Identity and Access Management resources.

Delete an unused access analyzer

Use the following procedure to delete an unused access analyzer. When you delete an analyzer, the resources are no longer monitored and no new findings are generated. All findings that were generated by the analyzer are deleted.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Under **Access analyzer**, choose **Unused access**.
3. Under **Access analyzer**, choose **Analyzer settings**.
4. In the **Analyzers** section, choose the name of the unused access analyzer to delete.
5. Choose **Delete analyzer**.
6. Enter **delete** and choose **Delete** to confirm deleting the analyzer.

View the IAM Access Analyzer findings dashboard

AWS Identity and Access Management Access Analyzer organizes external, internal, and unused access findings into a visual summary dashboard. The dashboard helps you gain visibility into the effective use of permissions at scale and identify accounts and AWS resources that need attention. You can use the dashboard to review findings by AWS organization, account, and finding type.

For external and internal access findings:

- The dashboard highlights the split between public access findings, external access findings, and internal access findings.
- The dashboard provides a breakdown of findings by resource type.

For unused access findings:

- The dashboard highlights the AWS accounts with the most unused access findings.
- The dashboard provides a breakdown of findings by type.

After you create any type of access analyzer, IAM Access Analyzer automatically adds new findings to the relevant dashboard. This allows you to identify and prioritize the areas with the most security concerns.

The summary dashboards give you a high-level view of the access issues detected by IAM Access Analyzer across your AWS environment. You can then drill down into the individual findings to investigate further and take appropriate actions to resolve them.

Viewing the summary dashboard for external and internal access analyzers

> **Note**
>
> After you create or update an analyzer, it can take time for the summary dashboard to reflect updates to findings.

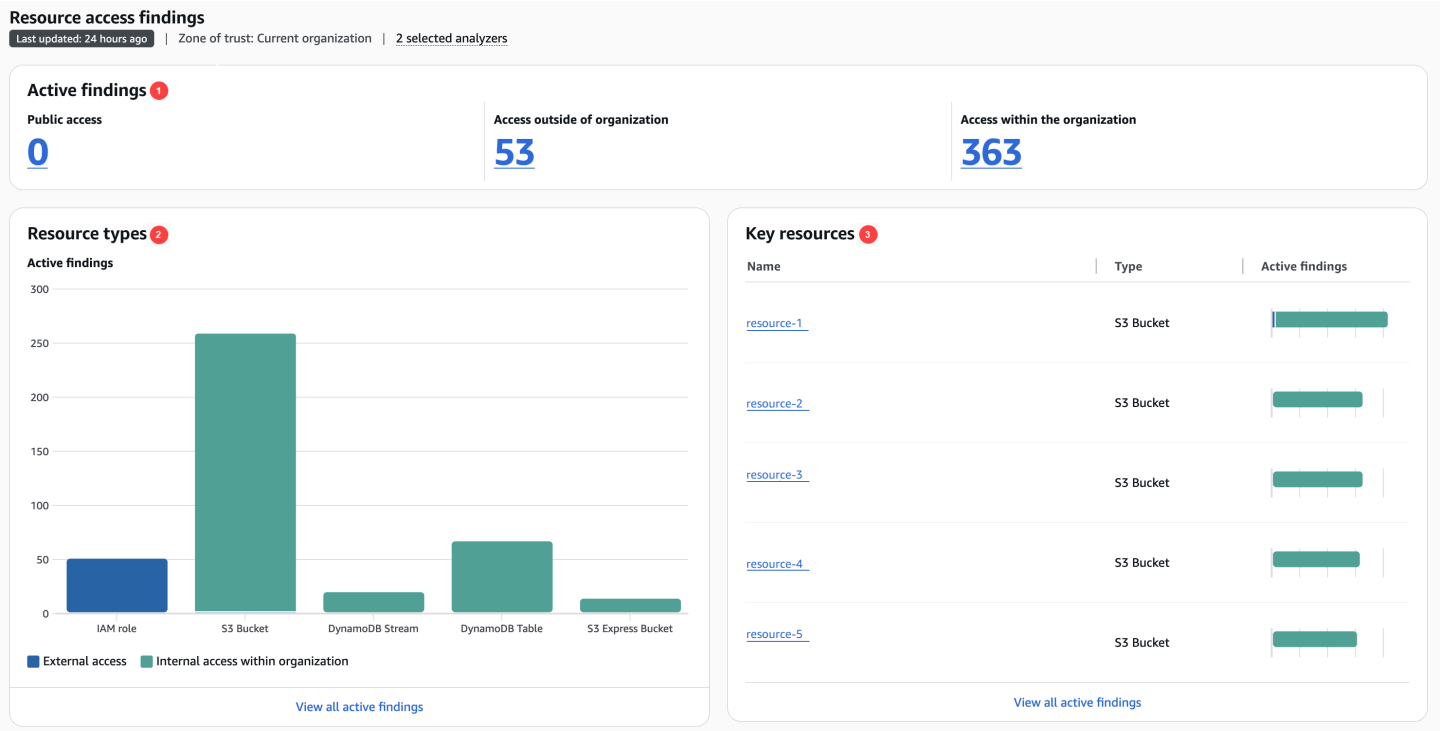
1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Access Analyzer**. The **Summary** window is displayed.
3. Choose **Select analyzers**.
4. In the **Select analyzers** window, choose **Organization** or **Account** for the **Zone of trust**.

   > **Note**
   >
   > Only the AWS Organizations management account or delegated administrator can choose **Organization** as the zone of trust.

5. Choose external and internal access analyzers from the **Resource access analyzers** dropdown.

   > **Note**
   >
   > You can select a maximum of one external access analyzer and a maximum of one internal access analyzer.

6. Choose **Update**. A summary of the findings for the selected external and internal access analyzers is displayed in the **Resource access findings** section.



In the preceding image, the resource findings dashboard is visible from within the **Summary** page.

1. The **Active findings** section includes the number of active findings for public access, the number of active findings that provide access outside of the account or organization, and the number of active internal access findings for the selected analyzers. Choose a number to list all of the active findings of each type.
2. The **Resource types** section includes a breakdown of the resource types with active findings for the selected analyzers. Choose **View all active findings** for a complete list of active findings for the selected analyzers.
3. The **Key resources** section includes a summary of the key resources with active findings. This information helps you prioritize findings for your business-critical resources. Choose **View all active findings** for a complete list of active findings for the selected analyzers.

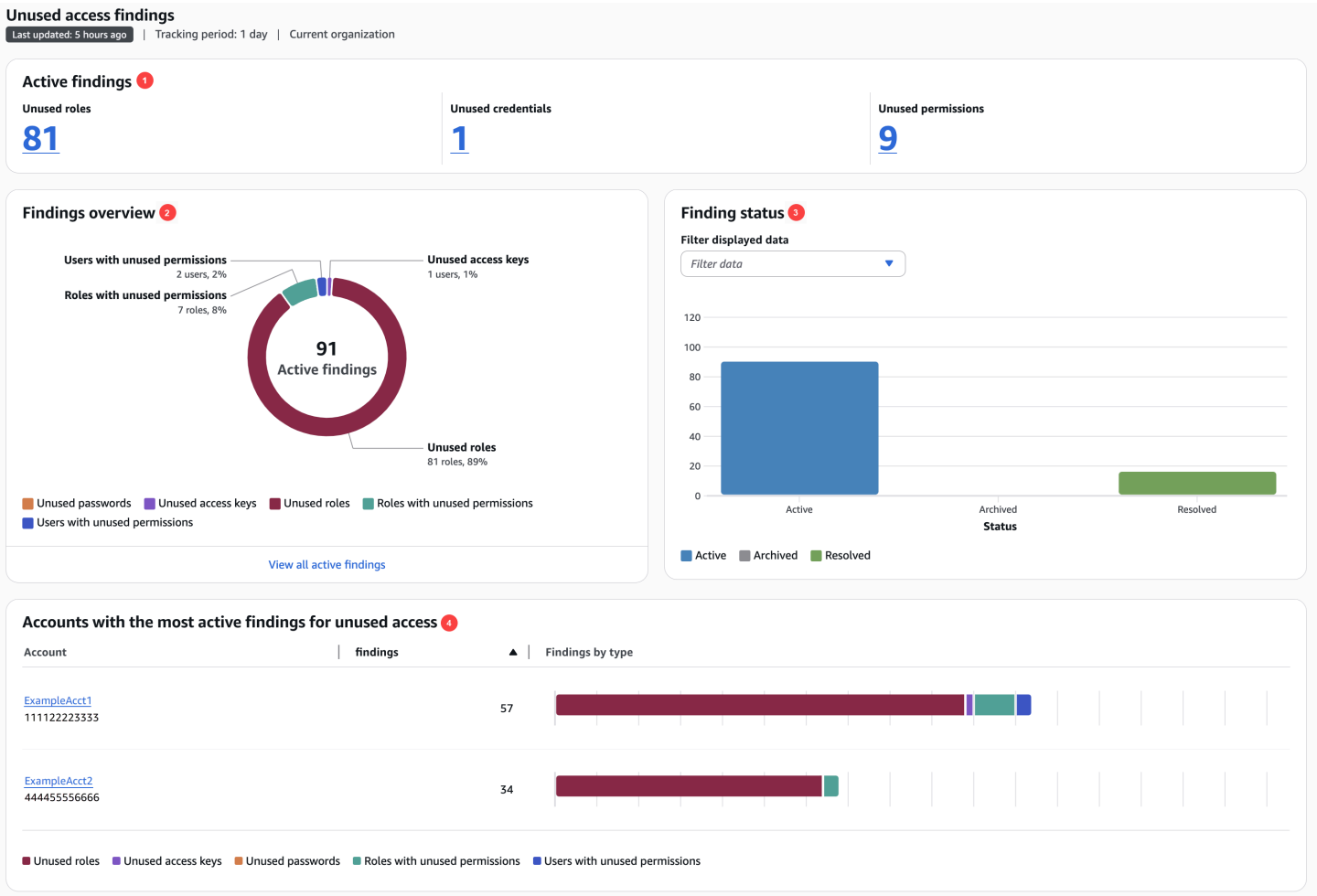
Viewing the summary dashboard for unused access analyzers

> **Note**
>
> After you create or update an analyzer, based on the amount of users and roles, it can take time for the summary dashboard to reflect updates to findings.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Access Analyzer**. The **Access Analyzer Summary** window is displayed.
3. Choose **Select analyzers**.
4. In the **Select analyzers** window, choose **Organization** or **Account** for the **Zone of trust**.

> **Note**
>
> Only the AWS Organizations management account or delegated administrator can choose **Organization** as the zone of trust.

5. Choose an unused access analyzer from the **Unused access analyzers** dropdown.
6. Choose **Update summary**. A summary of the findings for the selected unused access analyzer is displayed in the **Unused access findings** section.



In the preceding image, the unused access findings dashboard is visible from within the **Summary** page.

1. The **Active findings** section includes the number of active findings for unused roles, unused credentials, and unused permissions in your account or organization. **Unused credentials** include both unused access key and unused password findings. **Unused permissions** include both users and roles with unused permissions. Choose a number to list all of the active findings of each type.
2. The **Findings overview** section includes a breakdown of the type of active findings. Choose **View all active findings** for a complete list of active findings for the analyzer's account or organization.
3. The **Finding status** section includes a breakdown of the status of findings (**Active**, **Archived**, and **Resolved**) for your account or organization. You can select the findings statuses to display in the **Filter displayed data** dropdown.
4. The **Accounts with the most findings for unused access** section is only displayed if the selected accounts of your unused access analyzer is at the organization level. It includes a breakdown of the accounts in your organization with the most active findings. This is not an exhaustive list of every account in your organization. Your analyzer might have active findings for other accounts not listed in this section.

Review IAM Access Analyzer findings

After you enable IAM Access Analyzer, the next step is to review any findings to determine whether the access identified in the finding is intentional or unintentional. You can also review findings to determine similar findings for access that is intended, and then create an archive rule to automatically archive those findings. You can also review archived and resolved findings.

You should review all of the findings in your account to determine whether the external, internal, or unused access is expected and approved. If the access identified in the finding is expected, you can archive the finding. When you archive a finding, the status is changed to **Archived**, and the finding is removed from the active findings list. The finding is not deleted. You can view your archived findings at any time. Work through all of the findings in your account until you have zero active findings. After you get to zero findings, you know that any new **Active** findings that are generated are from a recent change in your environment.

> **Tip**
>
> If you have a large number of findings to review, use the IAM Access Analyzer API or AWS CLI commands to programmatically retrieve and manage your findings.

To review active findings for all types of access analyzers

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Access analyzer**. The findings dashboard is displayed.
3. Choose **Select analyzers**.
4. In the **Select analyzers** window, choose a maximum of one external access analyzer and a maximum of one internal access analyzer from the **Resource access analyzers** dropdown. Choose an unused access analyzer from the **Unused access analyzers** dropdown.
5. Choose **Update summary**. A summary of the active findings for the selected access analyzers is displayed on the dashboard. Choose a finding type in the **Resource access findings** or **Unused access findings** sections to view all active findings of the selected type.

   For more information on viewing the findings dashboard, see View the IAM Access Analyzer findings dashboard.

> **Note**
>
> Findings are displayed only if you have permission to view findings for the analyzer.

External and internal access findings

> **Note**
>
> IAM Access Analyzer charges for internal access analysis based on the number of resources monitored per Region per month. For more details about pricing, see IAM Access Analyzer pricing.

1. Under **Access Analyzer**, choose **Resource analysis**.

2. Choose **Select analyzers**.
3. In the **Select analyzers** window, choose a maximum of one external access analyzer and a maximum of one internal access analyzer from the **Resource access analyzers** dropdown.
4. Choose **Update summary**.

   The **Resource analysis** page displays the following details about the resources with active findings for the selected access analyzers:

Name

The name of the resource with active findings.

Type

The type of the resource.

Owner account

This column is displayed only if you are using an organization as the zone of trust for one or more of the selected analyzers. The account in the organization that owns the resource reported in the finding.

Active findings

A visual representation of the number and type of active findings for the resource. Hover over the field to display more information about the findings for the resource.

Public access

Indicates whether any of the findings for the resource allow public access.

Unused access findings

> **Note**
>
> IAM Access Analyzer charges for unused access analysis based on the number of IAM roles and users analyzed per month. For more details about pricing, see IAM Access Analyzer pricing.

1. Under **Access Analyzer**, choose **Unused access**.

2. Choose **Select analyzers**.
3. In the **Select analyzers** window, choose an unused access analyzer from the **Unused access analyzers** dropdown.
4. Choose **Update summary**.

The **Unused access** page displays the following details about the IAM entities that generated the findings for the selected access analyzer:

Finding ID

The unique ID assigned to the finding. Choose the finding ID to display additional details about the IAM entity that generated the finding.

Finding type

The type of unused access finding: **Unused access key**, **Unused password**, **Unused permission**, or **Unused role**.

IAM entity

The IAM entity reported in the finding. This can be an IAM user or role.

AWS account ID

This column is displayed only if you set up the analyzer for all AWS accounts in the organization. The AWS account in the organization that owns the IAM entity reported in the finding.

Last updated

The last time that the IAM entity reported in the finding was updated, or when the entity was created if no updates have been made.

Status

The status of the finding (**Active**, **Archived**, or **Resolved**).

Filter IAM Access Analyzer findings

The default filtering for a findings page is to display all active findings. To view all findings, choose **All** from the **Status** dropdown. To view archived findings, choose **Archived**. To view resolved findings, choose **Resolved**. When you first start using IAM Access Analyzer, there are no archived findings.

Use filters to display only the findings that meet the specified property criteria. To create a filter, select the property to filter on, then choose whether the property equals or contains a value, then enter or choose a property value to filter on.

For a list of filter keys that you can use to create or update an archive rule, see IAM Access Analyzer filter keys.

Filtering resources with active findings

You can view and filter active findings by resource for a maximum of one external access analyzer and a maximum of one internal access analyzer.

To filter resources with active findings

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Resource analysis**.
3. To filter by resource name, type all or part of the name of the resource in the search box.
4. In the **Filter access type** dropdown, choose the access type:
   - **All types** – display resources with all types of access findings.
   - **Public access** – display only resources with public access findings.
   - **External access** – display only resources with external access findings.
   - **Internal access within organization** – display only resources with internal access findings.
5. In the **Filter resource type** dropdown, choose a resource type to display only resources of the selected type.

Filtering external access findings

To filter external access findings

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Analyzer settings** and then choose the external access analyzer in the **Analyzers** section.
3. Choose **View findings**.
4. Choose the search box to display a list of available properties.
5. Choose the property to use to filter the findings displayed.

6. Choose the value to match for the property. Only findings with that value in the finding are displayed.

   For example, choose **Resource** as the property, then choose **Resource:**, then type part or all of the name of a bucket, then press Enter. Only findings for the bucket that matches the filter criteria are displayed. To create a filter that displays only findings for resources that allow public access, you can choose the **Public access** property, then choose **Public access =**, then choose **Public access = true**.

You can add additional properties to further filter the findings displayed. When you add additional properties, only findings that match all conditions in the filter are displayed. Defining a filter to display findings that match one property OR another property is not supported. Choose **Clear filters** to clear any filters you have defined and display all of the findings with the specified status for your analyzer.

Some fields are displayed only when you are viewing findings for an analyzer with an organization as its zone of trust.

The following properties are available for defining filters for external access:

- **Public access** – To filter by findings for resources that allow public access, filter by **Public access** then choose **Public access: true**.
- **Resource** – To filter by resource, type all or part of the name of the resource.
- **Resource Type** – To filter by resource type, choose the type from the list displayed.
- **Resource Owner Account** – Use this property to filter by the account in the organization that owns the resource reported in the finding.
- **Resource Control Policy Restriction** – Use this property to filter by the type of restriction applied by an Organizations resource control policy (RCP). To learn more, see Resource control policies (RCPs) in the AWS Organizations User Guide.
  - **Failed to evaluate RCP**: There was an error evaluating the RCP.
  - **Not applicable**: No RCP restricts this resource or principal. This also includes resources where RCPs are not yet supported.
  - **Applicable**: Your organization administrator has set restrictions through a RCP that impacts the resource or resource type. Contact your organization administrator for more details.
- **AWS Account** – Use this property to filter by AWS account that is granted access in the **Principal** section of a policy statement. To filter by AWS account, type all or part of the 12-digit AWS

account ID, or all or part of the full account ARN of the external AWS user or role that has access to resources in the current account.

- **Canonical User** – To filter by canonical user, type the canonical user ID as defined for Amazon S3 buckets. To learn more, see AWS Account Identifiers.
- **Federated User** – To filter by federated user, type all or part of the ARN of the federated identity. To learn more, see Identity Providers and Federation.
- **Finding ID** – To filter by finding ID, type all or part of the finding ID.
- **Error** – To filter by error type, choose **Access Denied** or **Internal Error**.
- **Principal ARN** – Use this property to filter on the ARN of the principal (IAM user, role, or group) used in an **aws:PrincipalArn** condition key. To filter by Principal ARN, type all or part of the ARN of the IAM user, role, or group from an external AWS account reported in a finding.
- **Principal OrgID** – To filter by Principal OrgID, type all or part of the organization ID associated with the external principals that belong to the AWS organization specified as a condition in the finding. To learn more, see AWS global condition context keys.
- **Principal OrgPaths** – To filter by Principal OrgPaths, type all or part of the ID for the AWS organization or organizational unit (OU) that allows access to all external principals that are account members of the specified organization or OU as a condition in the policy. To learn more, see AWS global condition context keys.
- **Source Account** – To filter on Source Account, type all or part of the AWS account ID associated with the resources, as used in some cross-service permissions in AWS. To learn more, see AWS global condition context keys.
- **Source ARN** – To filter by Source ARN, type all or part of the ARN specified as a condition in the finding. To learn more, see AWS global condition context keys.
- **Source IP** – To filter by Source IP, type all or part of the IP address that allows external entities access to resources in the current account when using the specified IP address. To learn more, see AWS global condition context keys.
- **Source OrgID** – To filter by Source OrgID, type all or part of the organization ID associated with the resources, as used in some cross-service permissions in AWS. To learn more, see AWS global condition context keys.
- **Source OrgPaths** – To filter by Source OrgPaths, type all or part of the organizational unit (OU) associated with the resources, as used in some cross-service permission in AWS. To learn more, see AWS global condition context keys.

- **Source VPC** – To filter by Source VPC, type all or part of the VPC ID that allows external entities access to resources in the current account when using the specified VPC. To learn more, see AWS global condition context keys.
- **Source VPC ARN** – To filter by Source VPC ARN, type all or part of the VPC ARN that allows external entities access to resources in the current account when using the specified VPC. To learn more, see AWS global condition context keys.
- **Source VPCE** – To filter by Source VPCE, type all or part of the VPC endpoint ID that allows external entities access to resources in the current account when using the specified VPC endpoint. To learn more, see AWS global condition context keys.
- **VPCE Account** – To filter by VPCE Account, type all or part of the 12-digit AWS account ID that owns the VPC endpoint external entities and allows external entities access to resources. To learn more, see AWS global condition context keys.
- **VPCE OrgID** – To filter by VPCE OrgID, type all or part of the organization ID that owns the VPC endpoint external entities and allows external entities access to resources. To learn more, see AWS global condition context keys.
- **VPCE OrgPaths** – To filter by VPCE OrgPaths, type all or part of the organizational unit (OU) that owns the VPC endpoint external entities and allows external entities access to resources. To learn more, see AWS global condition context keys.
- **User ID** – To filter by User ID, type all or part of the user ID of the IAM user from an external AWS account who is allowed access to resource in the current account. To learn more, see AWS global condition context keys.
- **KMS Key ID** – To filter by KMS key ID, type all or part of the key ID for the KMS key specified as a condition for AWS KMS-encrypted Amazon S3 object access in your current account.
- **Session Mode** – To filter by session mode for Amazon S3 directory buckets (`ReadOnly` or `ReadWrite`, type all or part of the session mode. To learn more, see CreateSession in the Amazon Simple Storage Service API Reference.
- **Google Audience** – To filter by Google Audience, type all or part of the Google application ID specified as a condition for IAM role access in your current account. To learn more, see IAM and AWS STS condition context keys.
- **Cognito Audience** – To filter by Amazon Cognito audience, type all or part of the Amazon Cognito identity pool ID specified as a condition for IAM role access in your current account. To learn more, see IAM and AWS STS condition context keys.

- **Caller Account** – The AWS account ID of the account that owns or contains the calling entity, such as an IAM role, user, or account root user. This is used by services calling AWS KMS. To filter by caller account, type all or part of the AWS account ID.
- **Facebook App ID** – To filter by Facebook App ID, type all or part of the Facebook application ID (or site ID) specified as a condition to allow Login with Facebook federation access to an IAM role in your current account. To learn more, see the **id** section in IAM and AWS STS condition context keys.
- **Amazon App ID** – To filter by Amazon App ID, type all or part of the Amazon application ID (or site ID) specified as a condition to allow Login with Amazon federation access to an IAM role in your current account. To learn more, see the **id** section in IAM and AWS STS condition context keys.
- **Lambda Event Source Token** – To filter on Lambda Event Source Token passed in with Alexa integrations, type all or part of the token string.

Filtering internal access findings

To filter internal access findings

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Analyzer settings** and then choose the internal access analyzer in the **Analyzers** section.
3. Choose **View findings**.
4. Choose the search box to display a list of available properties.
5. Choose the property to use to filter the findings displayed.
6. Choose the value to match for the property. Only findings with that value in the finding are displayed.

   For example, choose **Resource** as the property, then choose **Resource:**, then type part or all of the name of a bucket, then press Enter. Only findings for the bucket that matches the filter criteria are displayed.

You can add additional properties to further filter the findings displayed. When you add additional properties, only findings that match all conditions in the filter are displayed. Defining a filter to display findings that match one property OR another property is not supported. Choose **Clear**

filters to clear any filters you have defined and display all of the findings with the specified status for your analyzer.

Some fields are displayed only when you are viewing findings for an analyzer with an organization as its zone of trust.

The following fields are displayed only when you are viewing findings for an analyzer that is monitoring internal access:

- **Resource** – To filter by resource, type all or part of the name of the resource.
- **Resource Type** – To filter by resource type, choose the type from the list displayed.
- **Resource Owner Account** – Use this property to filter by the account in the organization that owns the resource reported in the finding.
- **Finding id** – To filter by finding ID, type all or part of the finding ID.

Filtering unused access findings

To filter unused access findings

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Unused access** and then choose the analyzer in the **View analyzer** dropdown.
3. Choose the search box to display a list of available properties.
4. Choose the property to use to filter the findings displayed.
5. Choose the value to match for the property. Only findings with that value in the finding are displayed.

   For example, choose **Findings type** as the property, then choose **Findings type =**, then choose **Findings type = Unused role**. Only findings with a type of **Unused role** are displayed.

You can add additional properties to further filter the findings displayed. When you add additional properties, only findings that match all conditions in the filter are displayed. Defining a filter to display findings that match one property OR another property is not supported. Choose **Clear filters** to clear any filters you have defined and display all of the findings with the specified status for your analyzer.

The following fields are displayed only when you are viewing findings for an analyzer that is monitoring unused access:

- **Findings type** – To filter by finding type, filter by **Findings type** and then choose the type of finding.
- **Resource** – To filter by resource, type all or part of the name of the resource.
- **Resource Type** – To filter by resource type, choose the type from the list displayed.
- **Resource Owner Account** – Use this property to filter by the account in the organization that owns the resource reported in the finding.
- **Finding id** – To filter by finding ID, type all or part of the finding ID.

Archive IAM Access Analyzer findings

When you get a finding for access to a resource that is intentional, you can archive the findings. For example, an external or internal access finding for an Amazon S3 bucket that is accessed for approved workflows or an unused access finding for an access key that may still be necessary. When you archive a finding, it is cleared from active findings list. Archived findings aren't deleted. You can filter the **Findings** page to display your archived findings, and unarchive them at any time.

To archive findings from the Findings page

1. Select the checkbox next to one or more findings to archive.
2. Choose **Actions** and then choose **Archive**.

   A confirmation is displayed at the top of the screen.

To archive findings from the Findings Details page

1. Choose the **Finding ID** for the finding to archive.
2. Choose **Archive**.

   A confirmation is displayed at the top of the screen.

To unarchive findings, repeat the preceding steps, but choose **Unarchive** instead of **Archive**. When you unarchive a finding, the status is set to Active.

Resolve IAM Access Analyzer findings

Resolving resource findings

To resolve external and internal access findings generated from unintended access, you should modify the policy statement to remove the permissions that allow access to the identified resource.

For findings related to Amazon S3 buckets, use the Amazon S3 console to configure the permissions on the bucket.

For IAM roles, use the IAM console to modify the trust policy for the listed IAM role.

For other supported resources, use the console to modify the policy statements that resulted in a generated finding.

After making a change to resolve a resource finding, such as modifying a policy applied to an IAM role, IAM Access Analyzer will scan the resource again. If the access to the resource is removed, the status of the finding is changed to **Resolved**. The finding will then be displayed in the resolved findings list instead of the active findings list.

> **Note**
>
> This does not apply to **Error** findings. When IAM Access Analyzer is not able to analyze a resource, it will generate an error finding. If you resolve the issue that prevented IAM Access Analyzer from analyzing the resource, the error finding will be removed completely instead of changing to a resolved finding. For more information, see IAM Access Analyzer error findings.

If the changes you made resulted in external or internal access to the resource, but in a different way, such as with a different principal or for a different permission, IAM Access Analyzer will resolve the original finding and generate a new **Active** finding. If the changes you made resulted in internal errors or access denied errors, all active non-error findings linked to the specific access of the resource are resolved and a new error finding is generated.

> **Note**
>
> For external access analyzers, it may take up to 30 minutes after a policy is modified for IAM Access Analyzer to analyze the resource again and then update the finding.

For internal access analyzers, it might take several minutes or hours for IAM Access Analyzer to analyze the resource again and then update the finding. IAM Access Analyzer automatically rescans all policies every 24 hours.
Resolved findings are deleted 90 days after the last update to the finding status.

Resolving unused access findings

IAM Access Analyzer provides recommended steps to resolve unused access analyzer findings based on the type of finding.

After you make a change to resolve an unused access finding, the status of the finding is changed to **Resolved** the next time the unused access analyzer runs. The finding is no longer displayed in the active findings list and instead is displayed in the resolved findings list. If you make a change that only partially addresses an unused access finding, the existing finding is changed to **Resolved** but a new finding is generated. For example, if you remove only some of the unused permissions in a finding, but not all of them.

IAM Access Analyzer charges for unused access analysis based on the number of IAM roles and users analyzed per month. For more details about pricing, see IAM Access Analyzer pricing.

Resolving unused permission findings

For unused permission findings, IAM Access Analyzer can recommend policies to remove from an IAM user or role and provide new policies to replace existing permissions policies. Policy recommendation is not supported for the following scenarios:

- The unused permission finding is for an IAM user that is in a user group.
- The unused permission finding is for an IAM role for IAM Identity Center.
- The unused permission finding has an existing permissions policy that includes the `notAction` element.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Unused access**.
3. Choose a finding with the **Finding type** of **Unused permissions**.
4. In the **Recommendations** section, if there are policies listed in the **Recommended policy** column, choose **Preview policy** to view the existing policy with the recommended policy to

replace the existing policy. If there are multiple recommended policies, you can choose **Next policy** and **Previous policy** to view each existing and recommended policy.

5. Choose **Download JSON** to download a .zip file with JSON files of all the recommended policies.
6. Create and attach the recommended policies to the IAM user or role. For more information, see Changing permissions for a user (console) and Modifying a role permissions policy (console).
7. Remove the policies listed in the **Existing permissions policy** column from the IAM user or role. For more information, see Removing permissions from a user (console) and Modifying a role permissions policy (console).

Resolving unused role findings

For unused role findings, IAM Access Analyzer recommends deleting the unused IAM role.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Unused access**.
3. Choose a finding with the **Finding type** of **Unused role**.
4. In the **Recommendations** section, review the details of the IAM role.
5. Delete the IAM role. For more information, see Deleting an IAM role (console).

Resolving unused access key findings

For unused access key findings, IAM Access Analyzer recommends deactivating or deleting the unused access key.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Unused access**.
3. Choose a finding with the **Finding type** of **Unused access keys**.
4. In the **Recommendations** section, review the details of the access key.
5. Deactivate or delete the access key. For more information, see Managing access keys (console).

Resolving unused password findings

For unused password findings, IAM Access Analyzer recommends deleting the unused password for the IAM user.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Unused access**.
3. Choose a finding with the **Finding type** of **Unused password**.
4. In the **Recommendations** section, review the details of the IAM user.
5. Delete the password for the IAM user. For more information, see Creating, changing, or deleting an IAM user password (console).

IAM Access Analyzer error findings

When IAM Access Analyzer analyzes resources, it typically generates findings that show who has access to your resources. However, in some cases, the analyzer might encounter issues that prevent it from completing the analysis. In these situations, IAM Access Analyzer generates error findings instead.

Error findings indicate that IAM Access Analyzer couldn't complete the analysis for a specific resource or for a specific principal-resource pair. These findings help you identify resources that might need attention to ensure proper analysis.

External access error findings

External access analyzers, which identify resources shared outside your account or organization, can generate two types of error findings:

- INTERNAL_ERROR – Indicates that IAM Access Analyzer encountered an internal issue while analyzing the resource. This could be due to service limitations or temporary issues.


```
{
 "findingDetails": [
  {
   "externalAccessDetails": {}
  }
 ],
 "resource": "arn:aws:iam::941407043048:role/TestAccessAnalyzer",
 "status": "ACTIVE",
 "error": "INTERNAL_ERROR",
 "createdAt": "2022-07-14T01:31:43.085000+00:00",
 "resourceType": "AWS::IAM::Role",
 "findingType": "ExternalAccess",
```

```
 "resourceOwnerAccount": "941407043048",
 "analyzedAt": "2025-03-19T06:51:46.109000+00:00",
 "id": "4b035c7d-b7d2-40e4-a6c3-9887d1a995df",
 "updatedAt": "2022-07-14T01:31:43.085000+00:00"
}
```


- ACCESS_DENIED – Indicates that IAM Access Analyzer doesn't have the required permissions to analyze the resource. This typically happens when the service-linked role (SLR) for IAM Access Analyzer is denied access to the resource.


```
{
 "findingDetails": [
  {
   "externalAccessDetails": {}
  }
 ],
 "resource": "arn:aws:kms:us-west-2:941407043048:key/01cae123-
b7f2-4488-9a05-0070a072ea2c",
 "status": "ACTIVE",
 "error": "ACCESS_DENIED",
 "createdAt": "2022-07-14T01:31:43.104000+00:00",
 "resourceType": "AWS::KMS::Key",
 "findingType": "ExternalAccess",
 "resourceOwnerAccount": "941407043048",
 "analyzedAt": "2025-03-19T06:51:46.090000+00:00",
 "id": "7ef6f04a-9d2c-4038-9cc0-2a5f00a4d8f8",
 "updatedAt": "2022-07-14T01:31:43.104000+00:00"
}
```


Internal access error findings

Internal access analyzers, which identify access within your account or organization, can generate four types of error findings:

- PRINCIPAL_LIMIT_EXCEEDED – Generated when more than 3,000 principals have access to a critical resource. This error helps you identify resources with overly broad access that might need to be restricted.

  If you make changes to the resource or principals in your environment that bring the number of principals below the limit, the analyzer will generate normal findings during the next scan, and the error finding will be marked as resolved.

```
{
 "id": "efec28fe-b304-412f-af0f-704d0d70c79c",
 "status": "ACTIVE",
 "error": "PRINCIPAL_LIMIT_EXCEEDED",
 "resource": "arn:aws:s3:::critical-data",
 "resourceType": "AWS::S3::Bucket",
 "resourceOwnerAccount": "111122223333",
 "createdAt": "2023-11-30T00:56:56.437000+00:00",
 "analyzedAt": "2024-03-06T04:11:54.406000+00:00",
 "updatedAt": "2023-11-30T00:56:56.437000+00:00",
 "findingType": "InternalAccess",
 "findingDetails": [
  {
   "internalAccessDetails": {}
  }
 ]
}
```

- Resource-level errors (INTERNAL_ERROR or ACCESS_DENIED) – Similar to external access errors, these indicate that the analyzer couldn't analyze a specific resource due to internal issues or permission problems. When a resource-level error occurs, the analyzer generates a single error finding for the resource instead of normal findings.

```
{
 "id": "efec28fe-b304-412f-af0f-704d0d70c79c",
 "status": "ACTIVE",
 "error": "INTERNAL_ERROR", // can be INTERNAL_ERROR or ACCESS_DENIED
 "resource": "arn:aws:s3:::critical-data",
 "resourceType": "AWS::S3::Bucket",
 "resourceOwnerAccount": "111122223333",
 "createdAt": "2023-11-30T00:56:56.437000+00:00",
 "analyzedAt": "2024-03-06T04:11:54.406000+00:00",
 "updatedAt": "2023-11-30T00:56:56.437000+00:00",
 "findingType": "InternalAccess",
 "findingDetails": [
  {
   "internalAccessDetails": {}
  }
 ]
}
```

- Principal-level errors (INTERNAL_ERROR or ACCESS_DENIED) – Indicates that the analyzer couldn't analyze access for a specific principal to a specific resource. Unlike resource-level errors, a resource can have both normal findings for some principals and error findings for other principals.

```
{
 "id": "efec28fe-b304-412f-af0f-704d0d70c79c",
 "status": "ACTIVE",
 "error": "INTERNAL_ERROR", // can be INTERNAL_ERROR or ACCESS_DENIED
 "resource": "arn:aws:s3:::critical-data",
 "resourceType": "AWS::S3::Bucket",
 "resourceOwnerAccount": "111122223333",
 "createdAt": "2023-11-30T00:56:56.437000+00:00",
 "analyzedAt": "2024-03-06T04:11:54.406000+00:00",
 "updatedAt": "2023-11-30T00:56:56.437000+00:00",
 "findingType": "InternalAccess",
 "findingDetails": [
  {
   "internalAccessDetails": {
    "principal": {
     "AWS": "arn:aws:iam::111122223333:role/MyRole_1"
    },
    "principalOwnerAccount": "111122223333",
    "principalType": "IAM_ROLE",
    "accessType": "INTRA_ACCOUNT"
   }
  }
 ]
}
```

- PRINCIPAL_ERRORS_LIMIT_EXCEEDED – Generated when there are too many principal-level error findings for a single resource. This is a resource-level error finding that may appear alongside normal findings for the same resource.

```
{
 "id": "efec28fe-b304-412f-af0f-704d0d70c79c",
 "status": "ACTIVE",
 "error": "PRINCIPAL_ERRORS_LIMIT_EXCEEDED",
 "resource": "arn:aws:s3:::critical-data",
 "resourceType": "AWS::S3::Bucket",
 "resourceOwnerAccount": "111122223333",
 "createdAt": "2023-11-30T00:56:56.437000+00:00",
```

```
 "analyzedAt": "2024-03-06T04:11:54.406000+00:00",
 "updatedAt": "2023-11-30T00:56:56.437000+00:00",
 "findingType": "InternalAccess",
 "resourceControlPolicyRestriction": "NOT_APPLICABLE",
 "serviceControlPolicyRestriction": "NOT_APPLICABLE",
 "findingDetails": [
  {
   "internalAccessDetails": {}
  }
 ]
}
```


Resolving error findings

If you resolve the issue that prevented IAM Access Analyzer from analyzing the resource, the error finding will be removed completely instead of changing to a resolved finding.

To resolve error findings, consider the following approaches based on the error type:

- For ACCESS_DENIED errors, verify that the IAM Access Analyzer service-linked role has the necessary permissions to access the resource.
- For PRINCIPAL_LIMIT_EXCEEDED errors, review the resource's access policies and consider restricting access to fewer principals.
- For INTERNAL_ERROR findings, you may need to wait for a subsequent analysis cycle or contact AWS support if the issue persists.
- For PRINCIPAL_ERRORS_LIMIT_EXCEEDED, review and potentially simplify the access patterns for the affected resource.

After making changes to address the underlying issues, IAM Access Analyzer will attempt to analyze the resources again during its next scan cycle.

IAM Access Analyzer supported resource types for external and internal access

For external and internal access analyzers, IAM Access Analyzer analyzes the resource-based policies that are applied to AWS resources in the Region where you enabled IAM Access Analyzer. It only analyzes resource-based policies. For details about how IAM Access Analyzer generates findings for each resource type, review the resource type information.

Note

The supported resource types listed are for external and internal access analyzers. Internal access analyzers don't support all resource types that external access analyzers support. Unused access analyzers only support IAM users and roles. For more information, see Understand how IAM Access Analyzer findings work.

Supported resource types for external access

- Amazon Simple Storage Service buckets
- Amazon Simple Storage Service directory buckets
- AWS Identity and Access Management roles
- AWS Key Management Service keys
- AWS Lambda functions and layers
- Amazon Simple Queue Service queues
- AWS Secrets Manager secrets
- Amazon Simple Notification Service topics
- Amazon Elastic Block Store volume snapshots
- Amazon Relational Database Service DB snapshots
- Amazon Relational Database Service DB cluster snapshots
- Amazon Elastic Container Registry repositories
- Amazon Elastic File System file systems
- Amazon DynamoDB streams
- Amazon DynamoDB tables

Supported resource types for internal access

- Amazon Simple Storage Service buckets
- Amazon Simple Storage Service directory buckets
- Amazon Relational Database Service DB snapshots
- Amazon Relational Database Service DB cluster snapshots

- Amazon DynamoDB streams
- Amazon DynamoDB tables

Amazon Simple Storage Service buckets

When IAM Access Analyzer analyzes Amazon S3 buckets for external access analyzers, it generates a finding when an Amazon S3 bucket policy, access control list (ACL), or access point, including a multi-Region access point, applied to a bucket grants access to an external entity. An external entity is a principal or other entity that you can use to create a filter that isn't within your zone of trust. For example, if a bucket policy grants access to another account or allows public access, IAM Access Analyzer generates a finding. However, if you enable Block Public Access on your bucket, you can block access at the account level or the bucket level.

For internal access analyzers, IAM Access Analyzer generates a finding when a principal (user or role) within your organization or account has access to a specified Amazon S3 bucket.

> **Note**
>
> IAM Access Analyzer doesn't analyze the access point policy attached to cross-account access points because the access point and its policy are outside the analyzer account. IAM Access Analyzer generates a public finding when a bucket delegates access to a cross-account access point and Block Public Access is not enabled on the bucket or account. When you enable Block Public Access, the public finding is resolved and IAM Access Analyzer generates a cross-account finding for the cross-account access point.

Amazon S3 *Block Public Access* settings override the bucket policies applied to the bucket. The settings also override the access point policies applied to the bucket's access points. IAM Access Analyzer analyzes Block Public Access settings at the bucket level whenever a policy changes. However, it evaluates the Block Public Access settings at the account level only once every 6 hours. This means that IAM Access Analyzer might not generate or resolve a finding for public access to a bucket for up to 6 hours. For example, if you have a bucket policy that allows public access, IAM Access Analyzer generates a finding for that access. If you then enable Block Public Access to block all public access to the bucket at the account level, IAM Access Analyzer doesn't resolve the finding for the bucket policy for up to 6 hours, even though all public access to the bucket is blocked. Resolution of public findings for cross-account access points can also take up to 6 hours once you enable Block Public Access at the account level. Changes to a resource control policy (RCP) without

a change to the bucket policy do not trigger a rescan of the bucket reported in the finding. IAM Access Analyzer analyzes the new or updated policy during the next periodic scan, which is within 24 hours.

For a multi-Region access point, IAM Access Analyzer uses an established policy for generating findings. IAM Access Analyzer evaluates changes to multi-Region access points once every 6 hours. This means IAM Access Analyzer doesn't generate or resolve a finding for up to 6 hours, even if you create or delete a multi-Region access point, or update the policy for it.

Amazon Simple Storage Service directory buckets

Amazon S3 directory buckets organize data hierarchically into directories as opposed to the flat storage structure of general purpose buckets, which is recommended for performance-critical workloads or applications. For external access analyzers, IAM Access Analyzer analyzes the directory bucket policy, including condition statements in a policy, that allow an external entity to access a directory bucket.

For internal access analyzers, IAM Access Analyzer generates a finding when a principal (user or role) within your organization or account has access to a specified Amazon S3 directory bucket.

Amazon S3 directory buckets also support access points, which enforce distinct permissions and network controls for all requests made to the directory bucket through the access point. Each access point can have an access point policy that works in conjunction with the bucket policy that is attached to the underlying directory bucket. With access points for directory buckets, you can restrict access to specific prefixes, API actions, or a virtual private cloud (VPC).

> **Note**
>
> IAM Access Analyzer doesn't analyze the access point policy attached to cross-account access points because the access point and its policy are outside the analyzer account. IAM Access Analyzer generates a public finding when a bucket delegates access to a cross-account access point and Block Public Access is not enabled on the bucket or account. When you enable Block Public Access, the public finding is resolved and IAM Access Analyzer generates a cross-account finding for the cross-account access point.

For more information about Amazon S3 directory buckets, see Working with directory buckets in the Amazon Simple Storage Service User Guide.

AWS Identity and Access Management roles

For IAM roles, IAM Access Analyzer analyzes trust policies. In a role trust policy, you define the principals that you trust to assume the role. A role trust policy is a required resource-based policy that is attached to a role in IAM. IAM Access Analyzer generates findings for roles within the zone of trust that can be accessed by an external entity that is outside your zone of trust.

> **Note**
>
> An IAM role is a global resource. If a role trust policy grants access to an external entity, IAM Access Analyzer generates a finding in each enabled Region.

AWS Key Management Service keys

For AWS KMS keys, IAM Access Analyzer analyzes the key policies and grants applied to a key. IAM Access Analyzer generates a finding if a key policy or grant allows an external entity to access the key. For example, if you use the kms:CallerAccount condition key in a policy statement to allow access to all users in a specific AWS account, and you specify an account other than the current account (the zone of trust for the current analyzer), IAM Access Analyzer generates a finding. To learn more about AWS KMS condition keys in IAM policy statements, see AWS KMS Condition Keys.

When IAM Access Analyzer analyzes a KMS key it reads key metadata, such as the key policy and list of grants. If the key policy doesn't allow the IAM Access Analyzer role to read the key metadata, an Access Denied error finding is generated. For example, if the following example policy statement is the only policy applied to a key, it results in an Access denied error finding in IAM Access Analyzer.

```
{
    "Sid": "Allow access for Key Administrators",
    "Effect": "Allow",
    "Principal": {
       "AWS": "arn:aws:iam::111122223333:role/Admin"
    },
    "Action": "kms:*",
    "Resource": "*"
}
```

Because this statement allows only the role named *Admin* from the AWS account 111122223333 to access the key, an Access Denied error finding is generated because IAM Access Analyzer isn't

able to fully analyze the key. An error finding is displayed in red text in the **Findings** table. The finding looks similar to the following.

```
{
    "error": "ACCESS_DENIED",
    "id": "12345678-1234-abcd-dcba-111122223333",
    "analyzedAt": "2019-09-16T14:24:33.352Z",
    "resource": "arn:aws:kms:us-
west-2:1234567890:key/1a2b3c4d-5e6f-7a8b-9c0d-1a2b3c4d5e6f7g8a",
    "resourceType": "AWS::KMS::Key",
    "status": "ACTIVE",
    "updatedAt": "2019-09-16T14:24:33.352Z"
}
```

When you create a KMS key, the permissions granted to access the key depend on how you create the key. If you receive an Access Denied error finding for a key resource, apply the following policy statement to the key resource to grant IAM Access Analyzer permission to access the key.

```
{
    "Sid": "Allow IAM Access Analyzer access to key metadata",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/aws-service-role/access-
analyzer.amazonaws.com/AWSServiceRoleForAccessAnalyzer"
        },
    "Action": [
        "kms:DescribeKey",
        "kms:GetKeyPolicy",
        "kms:List*"
    ],
    "Resource": "*"
},
```

After you receive an Access Denied finding for a KMS key resource, and then resolve the finding by updating the key policy, the finding is updated to a status of Resolved. If there are policy statements or key grants that grant permission to the key to an external entity, you might see additional findings for the key resource.

AWS Lambda functions and layers

For AWS Lambda functions, IAM Access Analyzer analyzes policies, including condition statements in a policy, that grant access to the function to an external entity. With Lambda, you can attach unique resource-based policies to functions, versions, aliases, and layers. IAM Access Analyzer reports external access based on resource-based policies attached to functions and layers. IAM Access Analyzer doesn't report external access based on resource-based policies attached to aliases and specific versions invoked using a qualified ARN.

For more information, see Using resource-based policies for Lambda and Using versions in the AWS Lambda Developer Guide.

Amazon Simple Queue Service queues

For Amazon SQS queues, IAM Access Analyzer analyzes policies, including condition statements in a policy, that allow an external entity access to a queue.

AWS Secrets Manager secrets

For AWS Secrets Manager secrets, IAM Access Analyzer analyzes policies, including condition statements in a policy, that allow an external entity to access a secret.

Amazon Simple Notification Service topics

IAM Access Analyzer analyzes resource-based policies attached to Amazon SNS topics, including condition statements in the policies that allow external access to a topic. You can allow external accounts to perform Amazon SNS actions such as subscribing to and publishing topics through a resource-based policy. An Amazon SNS topic is externally accessible if principals from an account outside of your zone of trust can perform operations on the topic. When you choose `Everyone` in your policy when creating an Amazon SNS topic, you make the topic accessible to the public. `AddPermission` is another way to add a resource-based policy to an Amazon SNS topic that allows external access.

Amazon Elastic Block Store volume snapshots

Amazon Elastic Block Store volume snapshots do not have resource-based policies. A snapshot is shared through Amazon EBS sharing permissions. For Amazon EBS volume snapshots, IAM Access Analyzer analyzes access control lists that allow an external entity access to a snapshot. An Amazon EBS volume snapshot can be shared with external accounts when encrypted. An unencrypted volume snapshot can be shared with external accounts and grant public access. Sharing settings

are in the `CreateVolumePermissions` attribute of the snapshot. When customers preview external access of an Amazon EBS snapshot, they can specify the encryption key as an indicator that the snapshot is encrypted, similar to how IAM Access Analyzer preview handles Secrets Manager secrets.

Amazon Relational Database Service DB snapshots

Amazon RDS DB snapshots do not have resource-based policies. A DB snapshot is shared through Amazon RDS database permissions, and only manual DB snapshots can be shared. For external access analyzers, IAM Access Analyzer analyzes access control lists that allow an external entity access to a Amazon RDS DB snapshot. Unencrypted DB snapshots can be public. Encrypted DB snapshots cannot be shared publicly, but they can be shared with up to 20 other accounts. For more information, see Creating a DB snapshot. IAM Access Analyzer considers the ability to export a database manual snapshot (for example, to an Amazon S3 bucket) as trusted access.

For internal access analyzers, IAM Access Analyzer generates a finding when a principal (user or role) within your organization or account has access to a specified Amazon RDS DB snapshot.

> **Note**
>
> IAM Access Analyzer does not identify public or cross-account access configured directly on the database itself. IAM Access Analyzer only identifies findings for public or cross-account access configured on the Amazon RDS DB snapshot.

Amazon Relational Database Service DB cluster snapshots

Amazon RDS DB cluster snapshots do not have resource-based policies. A snapshot is shared through Amazon RDS DB cluster permissions. For external access analyzers, IAM Access Analyzer analyzes access control lists that allow an external entity access to a Amazon RDS DB cluster snapshot. Unencrypted cluster snapshots can be public. Encrypted cluster snapshots cannot be shared publicly. Both unencrypted and encrypted cluster snapshots can be shared with up to 20 other accounts. For more information, see Creating a DB cluster snapshot. IAM Access Analyzer considers the ability to export a DB cluster snapshot (for example, to an Amazon S3 bucket) as trusted access.

For internal access analyzers, IAM Access Analyzer generates a finding when a principal (user or role) within your organization or account has access to a specified Amazon RDS DB cluster snapshot.

> **Note**
>
> IAM Access Analyzer findings do not include monitoring of any share of Amazon RDS DB clusters and clones with another AWS account or organization using AWS Resource Access Manager. IAM Access Analyzer only identifies findings for public or cross-account access configured on the Amazon RDS DB cluster snapshot.

Amazon Elastic Container Registry repositories

For Amazon ECR repositories, IAM Access Analyzer analyzes resource-based policies, including condition statements in a policy, that allow an external entity access to a repository (similar to other resource types like Amazon SNS topics and Amazon EFS file systems). For Amazon ECR repositories, a principal must have permission to `ecr:GetAuthorizationToken` through an identity-based policy to be considered externally available.

Amazon Elastic File System file systems

For Amazon EFS file systems, IAM Access Analyzer analyzes policies, including condition statements in a policy, that allow an external entity access to a file system. An Amazon EFS file system is externally accessible if principals from an account outside of your zone of trust can perform operations on that file system. Access is defined by a file system policy that uses IAM, and by how the file system is mounted. For example, mounting your Amazon EFS file system in another account is considered externally accessible, unless that account is in your organization and you have defined the organization as your zone of trust. If you are mounting the file system from a virtual private cloud with a public subnet, the file system is externally accessible. When you use Amazon EFS with AWS Transfer Family, file system access requests received from a Transfer Family server that is owned by a different account than the file system are blocked if the file system allows public access.

Amazon DynamoDB streams

For external access analyzers, IAM Access Analyzer generates a finding if a DynamoDB policy allows at least one cross-account action that allows an external entity to access a DynamoDB stream. For more information on the supported cross-account actions for DynamoDB, see IAM actions supported by resource-based policies in the Amazon DynamoDB Developer Guide.

For internal access analyzers, IAM Access Analyzer generates a finding when a principal (user or role) within your organization or account has access to a specified DynamoDB stream.

Amazon DynamoDB tables

For external access analyzers, IAM Access Analyzer generates a finding for a DynamoDB table if a DynamoDB policy allows at least one cross-account action that allows an external entity to access a DynamoDB table or index. For more information on the supported cross-account actions for DynamoDB, see IAM actions supported by resource-based policies in the Amazon DynamoDB Developer Guide.

For internal access analyzers, IAM Access Analyzer generates a finding when a principal (user or role) within your organization or account has access to a specified DynamoDB table.

Delegated administrator for IAM Access Analyzer

If you're configuring AWS Identity and Access Management Access Analyzer in your AWS Organizations management account, you can add a member account in the organization as the delegated administrator to manage IAM Access Analyzer for your organization. The delegated administrator has permissions to create and manage analyzers within the organization. Only the management account can add a delegated administrator.

The delegated administrator for IAM Access Analyzer is a member account within the organization that has permissions to create and manage analyzers that analyze access across the organization. Only the management account can add, remove, or change a delegated administrator.

If you add a delegated administrator, you can later change to a different account for the delegated administrator. When you do, the former delegated administrator account loses permission to all analyzers that were created using that account to analyze access across the organization. These analyzers move to a disabled state and no longer generate new or update existing findings. The existing findings for these analyzers are also no longer accessible. You can access them again in the future by configuring the account as the delegated administrator. If you know that you won't use the same account as a delegated administrator, consider deleting the analyzers before changing the delegated administrator. This deletes all findings generated. When the new delegated administrator creates new analyzers, new instances of the same findings are generated. You don't lose any findings, they just get generated for the new analyzer in a different account. And you can continue to access findings for the organization using the organization management account, which also has administrator permissions. The new delegated administrator must create new analyzers for IAM Access Analyzer to start monitoring resources in your organization.

If the delegated administrator leaves the AWS organization, the delegated administration privileges are removed from the account. All analyzers in the account with the organization as the

zone of trust move to a disabled state. The existing findings for these analyzers are also no longer accessible.

The first time that you configure analyzers in the management account, you can choose **Add delegated administrator** on the **Analyzer settings** page in the IAM Access Analyzer console.

> **Note**
>
> IAM Access Analyzer charges for unused access analyzers based on the number of IAM roles and users analyzed per analyzer per month. If you create an unused access analyzer in the management account and the delegated administrator account, you will be charged for both unused access analyzers. For more details about pricing, see IAM Access Analyzer pricing.

After you change the delegated administrator, the new administrator must create analyzers to start monitoring access to the resources in your organization.

Add a delegated administrator for IAM Access Analyzer

If you're configuring AWS Identity and Access Management Access Analyzer in your AWS Organizations management account, you can add a member account in the organization as the delegated administrator to manage IAM Access Analyzer for your organization. The delegated administrator has permissions to create and manage analyzers within the organization. Only the management account can add a delegated administrator.

To add a delegated administrator using the console

1. Log in to the AWS console using the management account for your organization.
2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. Under **Access Analyzer**, choose **Analyzer settings**.
4. Choose **Add delegated administrator**.
5. In the **Delegated administrator** field, enter the AWS account number of an organization member account to make the delegated administrator.

   The account must be a member of your organization.
6. Choose **Save changes**.

To add a delegated administrator using the AWS CLI or the AWS SDKs

When you create an analyzer to analyze access across the organization in a delegated administrator account using the AWS CLI, AWS API (using the AWS SDKs) or CloudFormation, you must use AWS Organizations APIs to enable service access for IAM Access Analyzer and register the member account as a delegated administrator.

1. Enable trusted service access for IAM Access Analyzer in AWS Organizations. See How to Enable or Disable Trusted Access in the AWS Organizations User Guide.
2. Register a valid member account of your AWS organization as a delegated administrator using the AWS Organizations `RegisterDelegatedAdministrator` API operation or the `register-delegated-administrator` AWS CLI command.

Archive rules

Archive rules automatically archive new findings that meet the criteria you define when you create the rule. You can also apply archive rules retroactively to archive existing findings that meet the archive rule criteria. For example, you can create an archive rule to automatically archive any findings for a specific Amazon S3 bucket that you regularly grant access to. Or if you grant access to multiple resources to a specific principal, you can create a rule that automatically archives any new finding generated for access granted to that principal. This lets you focus only on active findings that may indicate a security risk.

When you create an archive rule, only new findings that match the rule criteria are automatically archived. Existing findings are not automatically archived. When you create a rule, you can include up to 20 values per criterion in the rule. For a list of filter keys that you can use to create or update an archive rule, see IAM Access Analyzer filter keys.

> **Note**
>
> When you create or edit an archive rule, IAM Access Analyzer does not validate the values you include in the filter for the rule. For example, if you add a rule to match an AWS account, IAM Access Analyzer accepts any value in the field, even if it is not a valid AWS account number.

To create an archive rule

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Access analyzer**, then choose **Analyzer settings**.
3. In the **Analyzers** section, choose the analyzer for which you want to create an archive rule.
4. On the **Archive rules** tab, choose **Create archive rule**.
5. Enter a name for the rule if you want to change the default name.
6. In the **Rule** section, under **Criteria**, select a property to match for the rule.
7. Choose an condition for the property value, such as **Contains**, **Is**, or **Not Equals**.

   The operators available depend on the property you choose.
8. Optionally, add additional values for the property, or add additional criteria for the rule. For external access findings, to ensure that your rule won't archive new findings for public access, you can also include the criterion **Public access** and set it to **false**.

   To add another value for a criterion, choose **Add another value**. To add another criterion for the rule, choose **Add criterion**.
9. When you finish adding criteria and values, choose **Create rule** to apply the rule to new findings only. Choose **Create and archive active findings** to archive new and existing findings based on the rule criteria. In the **Results** section, you can review the list of active findings the archive rule applies to.

For example, to create a rule for external access findings that automatically archives any findings for Amazon S3 buckets: choose **Resource type**, and then choose **Is** for the condition. Next choose **S3 bucket** from the **Value** list.

To create a rule for unused access findings that automatically archives any finding for a particular account: choose **Resource Owner Account**, and then choose **Equals** for the condition. Type the AWS account ID in the **Value** text box.

Continue to define criteria to customize the rule as appropriate for your environment, and then choose **Create rule**.

If you are creating a new rule and add multiple criteria, you can remove a single criterion from the rule by choosing **Remove this criterion**. You can remove a value added for a criterion by choosing **Remove value**.

To edit an archive rule

1. Choose name of the rule to edit in the **Name** column.

   You can edit only one archive rule at a time.
2. Add new criteria or remove the existing criteria and values for each criterion.
3. Choose **Save changes** to apply the rule to new findings only. Choose **Save and archive active findings** to archive new and existing findings based on the rule criteria.

To delete an archive rule

1. Select the checkbox for the rules that you want to delete.
2. Choose **Delete**.
3. Type **`delete`** in the **Delete archive rule** confirmation dialog, and then choose **Delete**.

The rules are deleted only from the analyzer in the current Region. You must delete archive rules separately for each analyzer that you created in other Regions.

Monitoring AWS Identity and Access Management Access Analyzer with Amazon EventBridge

Use the information in this topic to learn how to monitor IAM Access Analyzer findings and access previews with Amazon EventBridge. EventBridge is the new version of Amazon CloudWatch Events.

Findings events

IAM Access Analyzer sends an event to EventBridge for each generated finding, for a change to the status of an existing finding, and when a finding is deleted. To receive findings and notifications about findings, you must create an event rule in Amazon EventBridge. When you create an event rule, you can also specify a target action to trigger based on the rule. For example, you could create an event rule that triggers an Amazon SNS topic when an event for a new finding is received from IAM Access Analyzer. Details about the resource control policy (RCP) are available in the event detail section.

Access preview events

IAM Access Analyzer sends an event to EventBridge for each access preview and change to its status. This includes an event when the access preview is first created (status Creating), when

the access preview is complete (status Completed), or when the access preview creation failed (status Failed). To receive notifications about access previews, you must create an event rule in EventBridge. When you create an event rule, you can specify a target action to trigger based on the rule. For example, you could create an event rule that triggers an Amazon SNS topic when an event for a completed access preview is received from IAM Access Analyzer.

Event notification frequency

IAM Access Analyzer sends events for new findings and findings with status updates to EventBridge within about an hour from when the event occurs in your account. IAM Access Analyzer also sends events to EventBridge when a resolved finding is deleted because the retention period has expired. For findings that are deleted because the analyzer that generated them is deleted, the event is sent to EventBridge approximately 24 hours after the analyzer was deleted. When a finding is deleted, the finding status is not changed. Instead, the `isDeleted` attribute is set to `true`. IAM Access Analyzer also sends events for newly created access previews and access preview status changes to EventBridge.

Example external access findings events

The following is an example IAM Access Analyzer external access finding event sent to EventBridge. The `id` listed is the ID for the event in EventBridge. To learn more, see Events and Event Patterns in EventBridge.

In the `detail` object, the values for the `accountId` and `region` attributes refer to the account and region reported in the finding. The `isDeleted` attribute indicates whether the event was from the finding being deleted. The `id` is the finding ID. The `resources` array is a singleton with the ARN of the analyzer that generated the finding.

```
{
    "account": "111122223333",
    "detail": {
        "accountId": "111122223333",
        "action": [
            "s3:GetObject"
        ],
        "analyzedAt": "2019-11-21T01:22:22Z",
        "condition": {},
        "createdAt": "2019-11-20T04:58:50Z",
        "id": "22222222-dcba-4444-dcba-333333333333",
        "isDeleted": false,
```

```
        "isPublic": false,
        "principal": {
            "AWS": "999988887777"
        },
        "region": "us-west-2",
        "resource": "arn:aws:s3:::amzn-s3-demo-bucket",
        "resourceType": "AWS::S3::Bucket",
        "status": "ACTIVE",
        "updatedAt": "2019-11-21T01:14:07Z",
        "version": "1.0"
    },
    "detail-type": "Access Analyzer Finding",
    "id": "11111111-2222-4444-aaaa-333333333333",
    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
    "source": "aws.access-analyzer",
    "time": "2019-11-21T01:22:33Z",
    "version": "0"
}
```

IAM Access Analyzer also sends events to EventBridge for error findings. An error finding is a finding generated when IAM Access Analyzer can't analyze the resource. Events for error findings include an `error` attribute as shown in the following example.

```
{
    "account": "111122223333",
    "detail": {
        "accountId": "111122223333",
        "analyzedAt": "2019-11-21T01:22:22Z",
        "createdAt": "2019-11-20T04:58:50Z",
        "error": "ACCESS_DENIED",
        "id": "22222222-dcba-4444-dcba-333333333333",
        "isDeleted": false,
        "region": "us-west-2",
        "resource": "arn:aws:s3:::amzn-s3-demo-bucket",
        "resourceType": "AWS::S3::Bucket",
        "status": "ACTIVE",
        "updatedAt": "2019-11-21T01:14:07Z",
        "version": "1.0"
    },
    "detail-type": "Access Analyzer Finding",
```

```
    "id": "11111111-2222-4444-aaaa-333333333333",
    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
    "source": "aws.access-analyzer",
    "time": "2019-11-21T01:22:33Z",
    "version": "0"
}
```

Example internal access findings events

The following is an example IAM Access Analyzer internal access finding event sent to EventBridge. The `id` listed is the ID for the event in EventBridge. To learn more, see Events and Event Patterns in EventBridge.

In the `detail` object, the values for the `accountId` and `principalOwnerAccount` attributes refer to the account of the principal reported in the finding. The `isDeleted` attribute indicates whether the event was from the finding being deleted. The `id` is the finding ID. The `resource` is the ARN of the analyzer that generated the finding.

```
{
    "version": "0",
    "id": "b45c3678-c278-b593-6121-c155259ce1b5",
    "detail-type": "Internal Access Finding",
    "source": "aws.access-analyzer",
    "account": "111122223333",
    "time": "2025-04-08T19:42:49Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:access-analyzer:us-east-1:111122223333:analyzer/testAnalyzer"
    ],
    "detail": {
        "accessType": "INTRA_ACCOUNT",
        "action": [
            "s3:GetObject"
        ],
        "analyzedAt": "2025-04-08T03:18:43.509465073Z",
        "condition": {},
        "createdAt": "2025-04-07T21:33:49.914099224Z",
        "id": "11111111-2222-4444-aaaa-333333333333",
        "isDeleted": false,
```

```
        "findingType": "InternalAccess",
        "principal": {
            "AWS": "arn:aws:iam::111122223333:role/MyRole_6"
        },
        "principalOwnerAccount": "111122223333",
        "principalType": "IAM_ROLE",
        "resource": "arn:aws:s3:::critical-data",
        "resourceControlPolicyRestrictionType": "NOT_APPLICABLE",
        "accountId": "111122223333",
        "resourceType": "AWS::S3::Bucket",
        "serviceControlPolicyRestrictionType": "NOT_APPLICABLE",
        "status": "ACTIVE",
        "updatedAt": "2025-04-08T03:22:12.654688231Z",
        "version": "1.0"
    }
}
```

IAM Access Analyzer also sends events to EventBridge for error findings. An error finding is a finding generated when IAM Access Analyzer can't analyze the resource. Events for error findings include an `error` attribute as shown in the following example.

```
{
    "version": "0",
    "id": "5a94b99b-e87d-a6a7-58c7-f47871532860",
    "detail-type": "Internal Access Finding",
    "source": "aws.access-analyzer-test",
    "account": "444455556666",
    "time": "2025-05-07T11:57:54Z",
    "region": "us-west-2",
    "resources": ["arn:aws:access-analyzer-beta:us-west-2:444455556666:analyzer/
example-analyzer"],
    "detail": {
        "analyzedAt": "2025-03-24T19:58:52.512329448Z",
        "createdAt": "2025-03-22T03:30:46.920200692Z",
        "id": "ef573afd-12a5-4095-87a6-bf2f25109895",
        "isDeleted": false,
        "findingType": "InternalAccess",
        "resource": "arn:aws:s3:::test-entity-88",
        "accountId": "111122223333",
        "resourceControlPolicyRestrictionType": "NOT_APPLICABLE",
        "resourceType": "AWS::S3::Bucket",
        "serviceControlPolicyRestrictionType": "NOT_APPLICABLE",
        "error": "ACCESS_DENIED", // can be INTERNAL_ERROR and ACCESS_DENIED
```

```
        "status": "ACTIVE",
        "updatedAt": "2025-03-24T20:09:39.176075014Z",
        "version": "1.0"
    }
}
```

Example unused access findings related events

The following is an example IAM Access Analyzer unused access finding event sent to EventBridge. The `id` listed is the ID for the event in EventBridge. To learn more, see Events and Event Patterns in EventBridge.

In the `detail` object, the values for the `accountId` and `region` attributes refer to the account and region reported in the finding. The `isDeleted` attribute indicates whether the event was from the finding being deleted. The `id` is the finding ID.

```
{
    "version": "0",
    "id": "dc7ce3ee-114b-3243-e249-7f10f9054b21",
    "detail-type": "Unused Access Finding for IAM entities",
    "source": "aws.access-analyzer",
    "account": "123456789012",
    "time": "2023-09-29T17:31:40Z",
    "region": "us-west-2",
    "resources": [
       "arn:aws:access-analyzer:us-west-2:123456789012:analyzer/
integTestLongLivingAnalyzer-DO-NOT-DELETE"
       ],
    "detail": {
        "findingId": "b8ae0460-5d29-4922-b92a-ba956c986277",
        "resource": "arn:aws:iam::111122223333:role/FindingIntegTestFakeRole",
        "resourceType": "AWS::IAM::Role",
        "accountId": "111122223333",
        "createdAt": "2023-09-29T17:29:18.758Z",
        "updatedAt": "2023-09-29T17:29:18.758Z",
        "analyzedAt": "2023-09-29T17:29:18.758Z",
        "previousStatus": "",
        "status": "ACTIVE",
        "version": "62160bda-8e94-46d6-ac97-9670930d8ffb",
        "isDeleted": false,
        "findingType": "UnusedPermission",
        "numberOfUnusedServices": 0,
        "numberOfUnusedActions": 1
```

```
        }
    }
```

IAM Access Analyzer also sends events to EventBridge for error findings. An error finding is a finding generated when IAM Access Analyzer can't analyze the resource. Events for error findings include an `error` attribute as shown in the following example.

```
{
    "version": "0",
    "id": "c2e7aa1a-4df7-7652-f33e-64113b8997d4",
    "detail-type": "Unused Access Finding for IAM entities",
    "source": "aws.access-analyzer",
    "account": "111122223333",
    "time": "2023-10-31T20:26:12Z",
    "region": "us-west-2",
    "resources": [
      "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/ba811f91-
de99-41a4-97c0-7481898b53f2"
      ],
    "detail": {
        "findingId": "b01a34f2-e118-46c9-aef8-0d8526b495c7",
        "resource": "arn:aws:iam::123456789012:role/TestRole",
        "resourceType": "AWS::IAM::Role",
        "accountId": "444455556666",
        "createdAt": "2023-10-31T20:26:08.647Z",
        "updatedAt": "2023-10-31T20:26:09.245Z",
        "analyzedAt": "2023-10-31T20:26:08.525Z",
        "previousStatus": "",
        "status": "ACTIVE",
        "version": "7c7a72a2-7963-4c59-ac71-f0be597010f7",
        "isDeleted": false,
        "findingType": "UnusedIAMRole",
        "error": "INTERNAL_ERROR"
        }
  }
```

Example access preview events

The following example shows data for the first event that is sent to EventBridge when you create an access preview. The `resources` array is a singleton with the ARN of the analyzer that the access preview is associated with. In the `detail` object, the `id` refers to the access preview ID and `configuredResources` refers to the resource for which the access preview was created.

The `status` is `Creating` and refers to the access preview status. The `previousStatus` is not specified because the access preview was just created.

```
{
    "account": "111122223333",
    "detail": {
        "accessPreviewId": "aaaabbbb-cccc-dddd-eeee-ffffaaaabbbb",
        "configuredResources": [
            "arn:aws:s3:::amzn-s3-demo-bucket"
        ],
        "createdAt": "2020-02-20T00:00:00.00Z",
        "region": "us-west-2",
        "status": "CREATING",
        "version": "1.0"
    },
    "detail-type": "Access Preview State Change",
    "id": "aaaabbbb-2222-3333-4444-555566667777",
    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
    "source": "aws.access-analyzer",
    "time": "2020-02-20T00:00:00.00Z",
    "version": "0"
}
```

The following example shows data for an event that is sent to EventBridge for an access preview with a status change from `Creating` to `Completed`. In the detail object, the `id` refers to the access preview ID. The `status` and `previousStatus` refer to the access preview status, where the previous status was `Creating` and the current status is `Completed`.

```
{
    "account": "111122223333",
    "detail": {
        "accessPreviewId": "aaaabbbb-cccc-dddd-eeee-ffffaaaabbbb",
        "configuredResources": [
            "arn:aws:s3:::amzn-s3-demo-bucket"
        ],
        "createdAt": "2020-02-20T00:00:00.000Z",
        "previousStatus": "CREATING",
        "region": "us-west-2",
        "status": "COMPLETED",
```

```
        "version": "1.0"
    },
    "detail-type": "Access Preview State Change",
    "id": "11112222-3333-4444-5555-666677778888",
    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
    "source": "aws.access-analyzer",
    "time": "2020-02-20T00:00:00.00Z",
    "version": "0"
}
```

The following example shows data for an event that is sent to EventBridge for an access preview with a status change from `Creating` to `Failed`. In the `detail` object, the `id` refers to the access preview ID. The `status` and `previousStatus` refer to the access preview status, where the previous status was `Creating` and the current status is `Failed`. The `statusReason` field provides the reason code indicating that the access preview failed due to an invalid resource configuration.

```
{
    "account": "111122223333",
    "detail": {
        "accessPreviewId": "aaaabbbb-cccc-dddd-eeee-ffffaaaabbbb",
        "configuredResources": [
            "arn:aws:s3:::amzn-s3-demo-bucket"
        ],
        "createdAt": "2020-02-20T00:00:00.00Z",
        "previousStatus": "CREATING",
        "region": "us-west-2",
        "status": "FAILED",
        "statusReason": {
            "code": "INVALID_CONFIGURATION"
        },
        "version": "1.0"
    },
    "detail-type": "Access Preview State Change",
    "id": "99998888-7777-6666-5555-444433332222",
    "region": "us-west-2",
    "resources": [
        "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/MyAnalyzer"
    ],
```

```
    "source": "aws.access-analyzer",
    "time": "2020-02-20T00:00:00.00Z",
    "version": "0"
}
```

Creating an event rule using the console

The following procedure describes how to create an event rule using the console.

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. Using the following values, create an EventBridge rule that monitors finding events or access preview events:
   - For **Rule type**, choose **Rule with an event pattern**.
   - For **Event source**, choose **Other**.
   - For **Event pattern**, choose **Custom patterns (JSON editor)**, and paste one of the following event pattern examples into the text area:
     - To create a rule based on any IAM Access Analyzer event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ]
}
```

     - To create a rule based on an external access, internal access, or unused access findings event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ],
  "detail-type": [
    "Access Analyzer Finding",
    "Internal Access Finding",
    "Unused Access Finding for IAM entities"
  ]
}
```

- To create a rule based only on an external access findings event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ],
  "detail-type": [
    "Access Analyzer Finding"
  ]
}
```

- To create a rule based only on an internal access findings event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ],
  "detail-type": [
    "Internal Access Finding"
  ]
}
```

- To create a rule based only on an unused access findings event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ],
  "detail-type": [
    "Unused Access Finding for IAM entities"
  ]
}
```

- To create a rule based on an access preview event, use the following pattern example:

```
{
  "source": [
    "aws.access-analyzer"
  ],
```

```
  "detail-type": [
    "Access Preview State Change"
  ]
}
```

- For **Target types**, choose **AWS service**, and for **Select a target**, choose a target such as an Amazon SNS topic or AWS Lambda function. The target is triggered when an event is received that matches the event pattern defined in the rule.

To learn more about creating rules, see Creating Amazon EventBridge rules that react to events in the *Amazon EventBridge User Guide*.

Creating an event rule using the CLI

1. Use the following to create a rule for Amazon EventBridge using the AWS CLI. Replace the rule name *TestRule* with the name for your rule.

```
aws events put-rule --name TestRule --event-pattern "{\"source\":[\"aws.access-analyzer\"]}"
```

2. You can customize the rule to trigger target actions only for a subset of generated findings, such as findings with specific attributes. The following example demonstrates how to create a rule that triggers a target action only for findings with a status of Active.

```
aws events put-rule --name TestRule --event-pattern "{\"source\":[\"aws.access-analyzer\"],\"detail-type\":[\"Access Analyzer Finding\"],\"detail\":{\"status\":[\"ACTIVE\"]}}"
```

   The following example demonstrates how to create a rule that triggers a target action only for access previews with a status from `Creating` to `Completed`.

```
aws events put-rule --name TestRule --event-pattern "{\"source\":[\"aws.access-analyzer\"],\"detail-type\":[\"Access Preview State Change\"],\"detail\":{\"status\":[\"COMPLETED\"]}}"
```

3. To define a Lambda function as a target for the rule you created, use the following example command. Replace the Region and the function name in the ARN as appropriate for your environment.

```
aws events put-targets --rule TestRule --targets Id=1,Arn=arn:aws:lambda:us-
east-1:111122223333:function:MyFunction
```

4. Add the permissions required to invoke the rule target. The following example demonstrates how to grant permissions to a Lambda function, following the preceding examples.

```
aws lambda add-permission --function-name MyFunction --statement-id 1 --action
 'lambda:InvokeFunction' --principal events.amazonaws.com
```

Integrate IAM Access Analyzer with AWS Security Hub CSPM

AWS Security Hub CSPM provides a comprehensive view of your security state across AWS. It helps you assess your environment against security industry standards and best practices. Security Hub CSPM collects security data from across AWS accounts, services, and supported third-party partner products. It then analyzes your security trends and identifies the highest priority security issues.

When you integrate IAM Access Analyzer with Security Hub CSPM, you can send findings from IAM Access Analyzer to Security Hub CSPM. Security Hub CSPM can then include those findings in its analysis of your overall security posture.

Contents

- How IAM Access Analyzer sends findings to Security Hub CSPM
  - Types of findings that IAM Access Analyzer sends
  - Latency for sending findings
  - Retrying when Security Hub CSPM is not available
  - Updating existing findings in Security Hub CSPM
- Viewing IAM Access Analyzer findings in Security Hub CSPM
  - Interpreting IAM Access Analyzer finding names in Security Hub CSPM
- Typical findings from IAM Access Analyzer
- Enabling and configuring the integration
- How to stop sending findings

How IAM Access Analyzer sends findings to Security Hub CSPM

In Security Hub CSPM, security issues are tracked as findings. Some findings come from issues that are detected by other AWS services or by third-party partners. Security Hub CSPM also has a set of rules that it uses to detect security issues and generate findings.

Security Hub CSPM provides tools to manage findings from across all of these sources. You can view and filter lists of findings and view detailed information about each finding. For more information, see Viewing findings in the *AWS Security Hub User Guide*. You can also track the status of investigations into findings. For more information, see Taking action on findings in the *AWS Security Hub User Guide*.

All findings in Security Hub CSPM use a standard JSON format called the AWS Security Finding Format (ASFF). The ASFF includes details about the source of the issue, the affected resources, and the current status of the finding. For more information, see AWS Security Finding Format (ASFF) in the *AWS Security Hub User Guide*.

AWS Identity and Access Management Access Analyzer is one of the AWS services that sends findings to Security Hub CSPM. For unused access, IAM Access Analyzer detects unused access granted to IAM users or roles and generates a finding for each of them. IAM Access Analyzer then sends these findings to Security Hub CSPM.

For external access, IAM Access Analyzer detects policy statements that allow public access or cross-account access to external principals on supported resources in your organization or account. IAM Access Analyzer generates a finding for public access and sends it to Security Hub CSPM. For cross-account access, IAM Access Analyzer sends a single finding for one external principal at a time to Security Hub CSPM. If there are multiple cross-account findings in IAM Access Analyzer, you must resolve the Security Hub CSPM finding for the single external principal before IAM Access Analyzer provides the next cross-account finding. For a full list of external principals with cross-account access outside the zone of trust for the analyzer, you must view the findings in IAM Access Analyzer. Details about the resource control policy (RCP) are available in the resource detail section.

Types of findings that IAM Access Analyzer sends

IAM Access Analyzer sends the findings to Security Hub CSPM using the AWS Security Finding Format (ASFF). In ASFF, the `Types` field provides the finding type. Findings from IAM Access Analyzer can have the following values for `Types`.

- External access findings – Effects/Data Exposure/External Access Granted

- External access findings – Software and Configuration Checks/AWS Security Best Practices/ External Access Granted
- Unused access findings – Software and Configuration Checks/AWS Security Best Practices/ Unused Permission
- Unused access findings – Software and Configuration Checks/AWS Security Best Practices/ Unused IAM Role
- Unused access findings – Software and Configuration Checks/AWS Security Best Practices/ Unused IAM User Password
- Unused access findings – Software and Configuration Checks/AWS Security Best Practices/ Unused IAM User Access Key

Latency for sending findings

When IAM Access Analyzer creates a new finding, it is usually sent to Security Hub CSPM within 30 minutes. However, there are rare cases when IAM Access Analyzer may not be notified about a policy change. For example:

- Changes to Amazon S3 account-level block public access settings can take up to 12 hours to be reflected in IAM Access Analyzer.
- Changes to a resource control policy (RCP) without a change to the resource-based policy do not trigger a rescan of the resource reported in the finding. IAM Access Analyzer analyzes the new or updated policy during the next periodic scan, which is within 24 hours.
- If there is a delivery issue with AWS CloudTrail log delivery, a policy change may not trigger a rescan of the resource that was reported in the finding.

In these situations, IAM Access Analyzer analyzes the new or updated policy during the next periodic scan.

Retrying when Security Hub CSPM is not available

If Security Hub CSPM is not available, IAM Access Analyzer retries sending the findings on a periodic basis.

Updating existing findings in Security Hub CSPM

After sending a finding to Security Hub CSPM, IAM Access Analyzer continues to send updates to reflect any additional observations of the finding activity to Security Hub CSPM. These updates are reflected within the same finding.

For external access findings IAM Access Analyzer groups them per resource. In Security Hub CSPM, the finding for a resource remains active if at least one of the findings for that resource is active in IAM Access Analyzer. If all findings in IAM Access Analyzer for a resource are archived or resolved, then the Security Hub CSPM finding is also archived. The Security Hub CSPM finding is updated when the policy access changes between public and cross-account access. This update can include changes to the type, title, description, and severity of the finding.

For unused access findings, IAM Access Analyzer does not group them by resource. Instead, if an unused access finding is resolved in IAM Access Analyzer, then the corresponding Security Hub CSPM finding is also resolved. The Security Hub CSPM finding is updated when you update the IAM user or role that generated the unused access finding.

Viewing IAM Access Analyzer findings in Security Hub CSPM

To view your IAM Access Analyzer findings in Security Hub CSPM, choose **See findings** in the **AWS: IAM Access Analyzer** section of the summary page. Alternatively, you can choose **Findings** from the navigation panel. You can then filter the findings to display only AWS Identity and Access Management Access Analyzer findings by choosing the **Product name:** field with a value of **`IAM Access Analyzer`**.

Interpreting IAM Access Analyzer finding names in Security Hub CSPM

AWS Identity and Access Management Access Analyzer sends the findings to Security Hub CSPM using the AWS Security Finding Format (ASFF). In ASFF, the **Types** field provides the finding type. ASFF types use a different naming scheme than AWS Identity and Access Management Access Analyzer. The following table includes details about all of the ASFF types associated with AWS Identity and Access Management Access Analyzer findings as they appear in Security Hub CSPM.

ASFF finding type	Security Hub CSPM finding title	Description
Effects/Data Exposure/ External Access Granted	<resource ARN> allows public access	A resource-based policy attached to the resource

ASFF finding type	Security Hub CSPM finding title	Description
		allows public access on the resource to all external principals.
Software and Configuration Checks/AWS Security Best Practices/External Access Granted	<resource ARN> allows cross-account access	A resource-based policy attached to the resource allows cross-account access to external principals outside the zone of trust for the analyzer.
Software and Configuration Checks/AWS Security Best Practices/Unused Permission	<resource ARN> contains unused permissions	A user or role contains unused service and action permissio ns.
Software and Configuration Checks/AWS Security Best Practices/Unused IAM Role	<resource ARN> contains unused IAM role	A user or role contains an unused IAM role.
Software and Configuration Checks/AWS Security Best Practices/Unused IAM User Password	<resource ARN> contains unused IAM user password	A user or role contains an unused IAM user password.
Software and Configuration Checks/AWS Security Best Practices/Unused IAM User Access Key	<resource ARN> contains unused IAM user access key	A user or role contains an unused IAM user access key.

Typical findings from IAM Access Analyzer

IAM Access Analyzer sends findings to Security Hub CSPM using the AWS Security Finding Format (ASFF).

Here is an example of a typical finding from IAM Access Analyzer for external access findings.

```
{
```

```
    "SchemaVersion": "2018-10-08",
    "Id": "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/my-analyzer/
arn:aws:s3:::amzn-s3-demo-bucket",
    "ProductArn": "arn:aws:securityhub:us-west-2::product/aws/access-analyzer",
    "GeneratorId": "aws/access-analyzer",
    "AwsAccountId": "111122223333",
    "Types": ["Software and Configuration Checks/AWS Security Best Practices/External
 Access Granted"],
    "CreatedAt": "2020-11-10T16:17:47Z",
    "UpdatedAt": "2020-11-10T16:43:49Z",
    "Severity": {
        "Product": 1,
        "Label": "LOW",
        "Normalized": 1
    },
    "Title": "AwsS3Bucket/arn:aws:s3:::amzn-s3-demo-bucket/ allows cross-account
 access",
    "Description": "AWS::S3::Bucket/arn:aws:s3:::amzn-s3-demo-bucket/ allows cross-
account access from AWS 444455556666",
    "Remediation": {
        "Recommendation": {"Text": "If the access isn't intended, it indicates a
 potential security risk. Use the console for the resource to modify or remove the
 policy that grants the unintended access. You can use the Rescan button on the Finding
 details page in the Access Analyzer console to confirm whether the change removed the
 access. If the access is removed, the status changes to Resolved."}
    },
    "SourceUrl": "https://console.aws.amazon.com/access-analyzer/home?region=us-
west-2#/findings/details/dad90d5d-63b4-6575-b0fa-ef9c556ge798",
    "Resources": [
        {
            "Type": "AwsS3Bucket",
            "Id": "arn:aws:s3:::amzn-s3-demo-bucket",
            "Details": {
                "Other": {
                    "External Principal Type": "AWS",
                    "Condition": "none",
                    "Action Granted": "s3:GetObject,s3:GetObjectVersion",
                    "External Principal": "444455556666"
                }
            }
        }
    ],
    "WorkflowState": "NEW",
    "Workflow": {"Status": "NEW"},
```

```
    "RecordState": "ACTIVE"
}
```

Here is an example of a typical finding from IAM Access Analyzer for unused access findings.

```
{
    "Findings": [
    {
      "SchemaVersion": "2018-10-08",
      "Id": "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/integTestAnalyzer-
DO-NOT-DELETE/arn:aws:iam::111122223333:role/TestRole/UnusedPermissions",
      "ProductArn": "arn:aws:securityhub:us-west-2::product/aws/access-analyzer",
      "ProductName": "IAM Access Analyzer",
      "CompanyName": "AWS",
      "Region": "us-west-2",
      "GeneratorId": "aws/access-analyzer",
      "AwsAccountId": "111122223333",
      "Types": [
        "Software and Configuration Checks/AWS Security Best Practices/Unused
 Permission"
      ],
      "CreatedAt": "2023-09-18T16:29:09.657Z",
      "UpdatedAt": "2023-09-21T20:39:16.651Z",
      "Severity": {
        "Product": 1,
        "Label": "LOW",
        "Normalized": 1
      },
      "Title": "AwsIamRole/arn:aws:iam::111122223333:role/IsengardRole-DO-NOT-DELETE/
 contains unused permissions",
      "Description": "AWS::IAM::Role/arn:aws:iam::111122223333:role/IsengardRole-DO-
NOT-DELETE/ contains unused service and action-level permissions",
      "Remediation": {
        "Recommendation": {
          "Text":"If the unused permissions aren’t required, delete the permissions to
 refine access to your account. Use the IAM console to modify or remove the policy that
 grants the unused permissions. If all the unused permissions are removed, the status
 of the finding changes to Resolved."
        }
      },
      "SourceUrl": "https://us-west-2.console.aws.amazon.com/access-analyzer/
home?region=us-west-2#/unused-access-findings?resource=arn%3Aaws%3Aiam%3A
%3A903798373645%3Arole%2FTestRole",
```

```
      "ProductFields": {
      "numberOfUnusedActions": "256",
      "numberOfUnusedServices": "15",
      "resourceOwnerAccount": "111122223333",
      "findingId": "DEMO24d8d-0d3f-4d3d-99f4-299fc8a62ee7",
      "findingType": "UnusedPermission",
      "aws/securityhub/FindingId": "arn:aws:securityhub:us-west-2::product/aws/access-
analyzer/arn:aws:access-analyzer:us-west-2:111122223333:analyzer/integTestAnalyzer-DO-
NOT-DELETE/arn:aws:iam::111122223333:role/TestRole/UnusedPermissions",
      "aws/securityhub/ProductName": "AM Access Analyzer",
      "aws/securityhub/CompanyName": "AWS"
    },
    "Resources": [
    {
      "Type": "AwsIamRole",
      "Id": "arn:aws:iam::111122223333:role/TestRole"
    }
  ],
  "WorkflowState": "NEW",
  "Workflow": {
    "Status": "NEW"
    },
  "RecordState": "ARCHIVED",
  "FindingProviderFields": {
    "Severity": {
      "Label": "LOW"
    },
    "Types": [
    "Software and Configuration Checks/AWS Security Best Practices/Unused Permission"
    ]
  }
  }
]
}
```


Enabling and configuring the integration

To use the integration with Security Hub CSPM, you must enable Security Hub CSPM. For information on how to enable Security Hub CSPM, see Setting up Security Hub in the *AWS Security Hub User Guide*.

When you enable both IAM Access Analyzer and Security Hub CSPM, the integration is enabled automatically. IAM Access Analyzer immediately begins to send findings to Security Hub CSPM.

How to stop sending findings

To stop sending findings to Security Hub CSPM, you can use either the Security Hub CSPM console or the API.

See Disabling and enabling the flow of findings from an integration (console) or Disabling the flow of findings from an integration (Security Hub API, AWS CLI) in the *AWS Security Hub User Guide*.

Logging IAM Access Analyzer API calls with AWS CloudTrail

IAM Access Analyzer is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in IAM Access Analyzer. CloudTrail captures all API calls for IAM Access Analyzer as events. The calls captured include calls from the IAM Access Analyzer console and code calls to the IAM Access Analyzer API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for IAM Access Analyzer. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**.

Using the information collected by CloudTrail, you can determine the request that was made to IAM Access Analyzer, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

IAM Access Analyzer information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in IAM Access Analyzer, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for IAM Access Analyzer, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations

- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All IAM Access Analyzer actions are logged by CloudTrail and are documented in the IAM Access Analyzer API Reference. For example, calls to the `CreateAnalyzer`, `CreateArchiveRule` and `ListFindings` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding IAM Access Analyzer log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateAnalyzer` operation made by an assumed-role session named `Alice-tempcreds` on "June 14, 2021". The role session was issued by the role named `admin-tempcreds`.

```
{
  "eventVersion": "1.05",
  "userIdentity":   {
    "type": "AssumedRole",
    "principalId": "AROAIBKEVSQ6C2EXAMPLE:Alice-tempcreds",
    "arn": "arn:aws:sts::111122223333:assumed-role/admin-tempcreds/Alice-tempcreds",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
```

```
      "attributes": {
        "mfaAuthenticated": "true",
        "creationDate": "2021-06-14T22:54:20Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/admin-tempcreds",
        "accountId": "111122223333",
        "userName": "admin-tempcreds"
      },
     "webIdFederationData": {},
    }
  },
  "eventTime": "2021-06-14T22:57:36Z",
  "eventSource": "access-analyzer.amazonaws.com",
  "eventName": "CreateAnalyzer",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "198.51.100.179",
  "userAgent": "aws-sdk-java/1.12.79 Linux/5.4.141-78.230 OpenJDK_64-
Bit_Server_VM/25.302-b08 java/1.8.0_302 vendor/Oracle_Corporation cfg/retry-mode/
standard",
  "requestParameters": {
    "analyzerName": "test",
    "type": "ACCOUNT",
    "clientToken": "11111111-abcd-2222-abcd-222222222222",
        "tags": {
            "tagkey1": "tagvalue1"
        }
  },
  "responseElements": {
    "arn": "arn:aws:access-analyzer:us-west-2:111122223333:analyzer/test"
  },
  "requestID": "22222222-dcba-4444-dcba-333333333333",
  "eventID": "33333333-bcde-5555-bcde-444444444444",
  "readOnly": false,
  "eventType": "AwsApiCall",,
  "managementEvent": true,
  "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

IAM Access Analyzer filter keys

You can use the filter keys below to define an archive rule (`CreateArchiveRule`), update an archive rule (`UpdateArchiveRule`), retrieve a list of findings (`ListFindings` and `ListFindingsV2`), or retrieve a list of access preview findings for a resource (`ListAccessPreviewFindings`). There is no difference between using IAM API and CloudFormation for configuring archive rules.

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
resource	**Resource**	The ARN uniquely identifying the resource that the external principal has access to. To learn more, see Amazon resource names (ARNs).	String	Yes	Yes	Yes	External Internal Unused
resourceType `AWS::S3::Bucket` | `AWS::IAM::Role` | `AWS::SQS::Queue` | `AWS::Lambda::Fun`	**Resource Type**	The type of resource that the external principal has access to. **Note** Internal access analyzers don't support all resource types that	String	Yes	Yes	Yes	External Internal Unused

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
ion | AWS::Lar da::Laye Version |AWS::KM :Key | AWS::Se etsMana r::Secr | AWS::EF :FileSy em | AWS::EC :Snapsh | AWS::EC :Reposi ry | AWS::RD :DBSnap ot | AWS::RD :DBClus rSnapsh | AWS::SN :Topic | AWS::S3		external access analyzers support. Unused access analyzers only support IAM users and roles. For more information, see IAM Access Analyzer supported resource types for external and internal access.					

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
`press::I rectoryI cket | AWS::Dyı moDB::Tı le | AWS::Dyı moDB::S eam | AWS::IAI :User`							
resourceO wnerAcco nt	**Resource Owner Account**	The 12 digit AWS account ID that owns the resource. To learn more, see AWS account identifiers.	String	✓ Yes	✓ Yes	✓ Yes	External Internal Unused
isPublic	**Public access**	Indicates whether the finding reports a resource that has a policy that allows public access.	Boolean	✓ Yes	✓ Yes	✓ Yes	External

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
findingType `ExternalAccess` | `UnusedIAMRole` | `UnusedIAMUserAccessKey` | `UnusedIAMUserPassword` | `UnusedPermission` | `InternalAccess`	**Findings type**	The type of the finding. For external access analyzers, the type is `ExternalAccess`. For unused access analyzers, the type can be `UnusedIAMRole`, `UnusedIAMUserAccessKey`, `UnusedIAMUserPassword`, or `UnusedPermission`. For internal access analyzers, the type is `InternalAccess`.	String	Yes	Yes	Yes	External Internal Unused

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
resourceControlPolicyRestriction APPLIED | APPLICABLE | FAILED_EVALUATING_RCP | NOT_APPLICABLE	**Resource control policy (RCP) restriction**	The type of restriction applied by the resource owner with an Organizations resource control policy (RCP). For more information about the values for this filter key, see ExternalAccessDetails and InternalAccessDetails in the IAM Access Analyzer API Reference.	String	Yes	Yes	Yes	External Internal
serviceControlPolicyRestriction APPLIED | APPLICABLE | FAILED_EVALUATING_SCP | NOT_APPLICABLE	**Service control policy (SCP) restriction**	The type of restriction applied by an Organizations service control policy (SCP). For more information about the values for this filter key, see InternalAccessDetails in the IAM Access Analyzer API Reference.	String	Yes	Yes	Yes	Internal

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
status `ACTIVE` | `ARCHIVED` | `RESOLVED`	**Status**	The current status of the finding.	String	⊗ No	⊙ Yes	⊙ Yes	External Internal Unused
error	**Error**	Indicates the error reported for the finding.	String	⊙ Yes	⊙ Yes	⊙ Yes	External Internal
principal.AWS	**AWS Account**	The account granted access to the resource in the `Principal` field of the finding. Enter the 12-digit AWS account ID or the ARN of the external AWS user or role. To learn more, see AWS account identifiers.	String	⊙ Yes	⊙ Yes	⊙ Yes	External

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
principal.Federated	**Federated User**	The ARN of the federated identity that has access to the resource in the finding. To learn more, see Identity providers and federation	String	Yes	Yes	Yes	External
condition.aws:PrincipalArn	**Principal ARN**	The ARN of the principal (IAM user, role, or group) indicated as the condition for resource access. To learn more, see AWS global condition context keys.	String	Yes	Yes	Yes	External
condition.aws:PrincipalOrgID	**Principal OrgID**	The organization identifier of the principal indicated as the condition for resource access. To learn more, see AWS global condition context keys.	String	Yes	Yes	Yes	External

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
condition.aws:PrincipalOrgPaths	**Principal OrgPaths**	The organization or organizational unit (OU) ID indicated as the condition for resource access. To learn more, see AWS global condition context keys.	String	Yes	Yes	Yes	External
condition.aws:SourceIp	**Source IP**	The IP address that allows the principal access to the resource when using the specified IP address. To learn more, see AWS global condition context keys.	IP address	Yes	Yes	Yes	External

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
condition .aws:SourceVpc	**Source VPC**	The VPC ID that allows the principal access to the resource when using the specified VPC. To learn more, see AWS global condition context keys.	String	Yes	Yes	Yes	External
condition .aws:UserId	**User ID**	The user ID of the IAM user from an external account indicated as the condition for access to the resource. To learn more, see AWS global condition context keys.	String	Yes	Yes	Yes	External
condition .aws:VpceAccount	**VPCE Account**	The account ID of the VPC endpoint that allows the principal access to the resource. To learn more, see AWS global condition context keys.	String	Yes	Yes	Yes	External Internal

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
condition.aws:SourceVpcArn	**Source VPC Arn**	The VPC ARN that allows the principal access to the resource when using the specified VPC. To learn more, see AWS global condition context keys.	ARN	Yes	Yes	Yes	External
condition.aws:VpceOrgID	**VPCE OrgID**	The organizational ID for the VPC endpoint that allows the principal access to the resource. To learn more, see AWS global condition context keys.	String	Yes	Yes	Yes	External Internal

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
condition .aws:VpceOrgPaths	**VPCE OrgPaths**	The organizational unit (OU) for the VPC endpoint that allows the principal access to the resource. To learn more, see AWS global condition context keys.	String (list)	Yes	Yes	Yes	External Internal
condition .cognito-identity.amazonaws.com:aud	**Cognito Audience**	The Amazon Cognito identity pool ID specified as a condition for IAM role access in the finding. To learn more, see IAM and AWS STS condition context keys.	String	Yes	Yes	Yes	External

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
condition.graph.facebook.com:app_id	**Facebook App ID**	The Facebook application ID (or site ID) specified as a condition to allow Login with Facebook federation access to the IAM role in the finding. To learn more, see IAM and AWS STS condition context keys.	String	Yes	Yes	Yes	External
condition.accounts.google.com:aud	**Google Audience**	The Google application ID specified as a condition for access to the IAM role. To learn more, see IAM and AWS STS condition context keys.	String	Yes	Yes	Yes	External

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
condition.kms:CallerAccoun	**KMS Key ID**	The AWS account ID that owns the calling entity (IAM user, role or account root user) used by services calling AWS KMS. To learn more, see Condition keys for AWS Key Management Service.	String	✓ Yes	✓ Yes	✓ Yes	External
condition.www.amazon.com:app_id	**Amazon App ID**	The Amazon application ID (or site ID) specified as a condition to allow Login with Amazon federation access to the role. To learn more, see	String	✓ Yes	✓ Yes	✓ Yes	External
id	**Finding ID**	The ID of the finding.	String	✗ No	✓ Yes	✓ Yes	External Internal Unused

Criterion	AWS Management Console field	Description	Type	Archive rule	List findings	List access preview findings	Supported analyzer types
changeType CHANGED | NEW | UNCHANGI		Provides context on how the access preview finding compares to existing access identified in IAM Access Analyzer.	String	⊗ No	⊗ No	⊙ Yes	External
existingFindingId		The existing ID of the finding in IAM Access Analyzer, provided only for existing findings in the access preview.	String	⊗ No	⊗ No	⊙ Yes	External
existingFindingStatus		The existing status of the finding, provided only for existing findings in the access preview.	String	⊗ No	⊗ No	⊙ Yes	External

Using service-linked roles for AWS Identity and Access Management Access Analyzer

AWS Identity and Access Management Access Analyzer uses an IAM service-linked role. A service-linked role is a unique type of IAM role linked directly to IAM Access Analyzer. Service-linked roles are predefined by IAM Access Analyzer and include all the permissions that the feature requires to call other AWS services on your behalf.

A service-linked role makes setting up IAM Access Analyzer easier because you don't have to manually add the necessary permissions. IAM Access Analyzer defines the permissions of its

service-linked roles, and unless defined otherwise, only IAM Access Analyzer can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS Identity and Access Management Access Analyzer

AWS Identity and Access Management Access Analyzer uses the service-linked role named **AWSServiceRoleForAccessAnalyzer** – Allow Access Analyzer to analyze resource metadata for external access and to analyze activity to identify unused access.

The AWSServiceRoleForAccessAnalyzer service-linked role trusts the following services to assume the role:

- `access-analyzer.amazonaws.com`

The role permissions policy named `AccessAnalyzerServiceRolePolicy` allows IAM Access Analyzer to complete actions on specific resources.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a service-linked role for IAM Access Analyzer

You don't need to manually create a service-linked role. When you enable Access Analyzer in the AWS Management Console or the AWS API, IAM Access Analyzer creates the service-linked role for you. The same service-linked role is used in all Regions in which you enable IAM Access Analyzer. Both external access and unused access findings use the same service-linked role.

> **Note**
>
> IAM Access Analyzer is Regional. You must enable IAM Access Analyzer in each Region independently.

If you delete this service-linked role, IAM Access Analyzer recreates the role when you next create an analyzer.

You can also use the IAM console to create a service-linked role with the **Access Analyzer** use case. In the AWS CLI or the AWS API, create a service-linked role with the `access-analyzer.amazonaws.com` service name. For more information, see Creating a Service-Linked Role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for IAM Access Analyzer

IAM Access Analyzer does not allow you to edit the AWSServiceRoleForAccessAnalyzer service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for IAM Access Analyzer

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that isn't actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

If IAM Access Analyzer is enabled in one or more regions in your AWS Organizations, you must delete all analyzers in all regions for your organization before attempting to delete this role.

> **Note**
>
> If IAM Access Analyzer is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete IAM Access Analyzer resources used by the AWSServiceRoleForAccessAnalyzer role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the **Access reports** section, under **Access analyzer**, choose **Analyzers**.
3. Choose the analyzer from which you want to delete IAM Access Analyzer resources attached to the service-linked role.

4. Choose **Delete**.
5. To confirm that you want to delete the analyzers, enter **`delete`**, and then choose **Delete**.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAccessAnalyzer service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for IAM Access Analyzer service-linked roles

IAM Access Analyzer supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS Regions and Endpoints.

Preview access

In addition to helping you identify resources that are shared with an external entity, AWS IAM Access Analyzer also shows you a preview of IAM Access Analyzer findings before deploying resource permissions so you can validate that your policy changes grant only intended public and cross-account access to your resource. This helps you start with intended external access to your resources.

You can preview and validate public and cross-account access to your Amazon S3 buckets in the Amazon S3 console. You can also use IAM Access Analyzer APIs to preview public and cross-account access for your Amazon S3 buckets, AWS KMS keys, IAM roles, Amazon SQS queues and Secrets Manager secrets by providing proposed permissions for your resource.

Topics

- Previewing access in Amazon S3 console
- Previewing access with IAM Access Analyzer APIs

Previewing access in Amazon S3 console

After you complete your bucket policy in the Amazon S3 console you have the option to preview public and cross-account access to your Amazon S3 bucket. You can validate that your policy changes grant only intended external access before you choose **Save changes**. This optional step enables you to preview AWS Identity and Access Management Access Analyzer findings for your

bucket. You can validate whether the policy change introduces new findings or resolves existing findings for external access. You can skip this validation step and save your Amazon S3 bucket policy at any time.

To preview external access to your bucket, you must have an active account analyzer in your bucket's region with the account as the zone of trust. You must also have the permissions required to use IAM Access Analyzer and preview access. For more information on enabling IAM Access Analyzer and permissions required, see Getting started with AWS Identity and Access Management Access Analyzer.

To preview access to your Amazon S3 bucket when you create or edit your bucket policy

1. Once you finish creating or editing your bucket policy, ensure your policy is a valid Amazon S3 bucket policy. The policy ARN must match the bucket ARN and the policy elements must be valid.
2. Below the policy, under **Preview external access**, choose an active account analyzer, then choose **Preview**. A preview of IAM Access Analyzer findings is generated for your bucket. The preview analyzes the displayed Amazon S3 bucket policy, together with the existing bucket permissions. This includes the bucket and account BPA settings, bucket ACL, the Amazon S3 access points and multi-region access points attached to the bucket, and their policies and BPA settings.
3. When the access preview completes, a preview of IAM Access Analyzer findings is displayed. Each finding reports an instance of a principal outside of the account with access to your bucket after you save the policy. You can validate access to your bucket by reviewing each finding. The finding header provides a summary of the access and you can expand the finding to review the finding details. Finding badges provide context on how saving the bucket policy would change access to the bucket. For example, they help you validate whether the policy change introduces new findings or resolves existing findings for external access:

    a. **New** – indicates a finding for new external access that the policy would introduce.

    b. **Resolved** – indicates a finding for existing external access that the policy would remove.

    c. **Archived** – indicates a finding for new external access that would be automatically archived, based on the archive rules for the analyzer that define when findings should be marked as intended.

    d. **Existing** – indicates an existing finding for external access that would remain unchanged.

    e. **Public** – if a finding is for public access to the resource, it will have a **Public** badge, in addition to one of the badges above.

4. If you identify external access you do not intend to introduce or remove, you can revise the policy and then choose **Preview** again until you have achieved the external access you intend. If you have a finding labeled **Public**, we recommend you revise the policy to remove public access before you choose **Save changes**. Previewing access is an optional step and you can choose **Save changes** at any time.

Previewing access with IAM Access Analyzer APIs

You can use IAM Access Analyzer APIs to preview public and cross-account access for your Amazon S3 buckets, AWS KMS keys, IAM roles, Amazon SQS queues and Secrets Manager secrets. You can preview access by providing proposed permissions for an existing resource you own or a new resource you want to deploy.

To preview external access to your resource, you must have an active account analyzer for the account and region of the resource. You must also have the permissions required to use IAM Access Analyzer and preview access. For more information on enabling IAM Access Analyzer and permissions required, see Getting started with AWS Identity and Access Management Access Analyzer.

To preview access for a resource, you can use the `CreateAccessPreview` operation and provide the analyzer ARN and the access control configuration for the resource. The service returns the unique ID for the access preview, which you can use to check the status of the access preview with the `GetAccessPreview` operation. When the status is `Completed`, you can use the `ListAccessPreviewFindings` operation to retrieve the findings generated for the access preview. The `GetAccessPreview` and `ListAccessPreviewFindings` operations will retrieve access previews and findings created within about 24 hours.

Each finding retrieved contains finding details describing the access. A preview status of the finding describing whether the finding would be `Active`, `Archived`, or `Resolved` after permissions deployment, and a `changeType`. The `changeType` provides context on how the access preview finding compares to existing access identified in IAM Access Analyzer:

- **New** – the finding is for newly introduced access.
- **Unchanged** – the preview finding is an existing finding that would remain unchanged.
- **Changed** – the preview finding is an existing finding with a change in status.

The `status` and the `changeType` help you understand how the resource configuration would change existing resource access. If the `changeType` is `Unchanged` or `Changed`, the finding will also contain the existing ID and status of the finding in IAM Access Analyzer. For example, a `Changed` finding with preview status `Resolved` and existing status `Active` indicates the existing `Active` finding for the resource would become `Resolved` as a result of the proposed permissions change.

You can use the `ListAccessPreviews` operation to retrieve a list of access previews for the specified analyzer. This operation will retrieve information on access preview created within about one hour.

In general, if the access preview is for an existing resource and you leave a configuration option unspecified, the access preview will use the existing resource configuration by default. If the access preview is for a new resource and you leave a configuration option unspecified, the access preview will use the default value depending on the resource type. For configuration cases for each resource type, see below.

Preview access to your Amazon S3 bucket

To create an access preview for a new Amazon S3 bucket or an existing Amazon S3 bucket that you own, you can propose a bucket configuration by specifying the Amazon S3 bucket policy, bucket ACLs, bucket BPA settings, and Amazon S3 access points, including multi-region access points, attached to the bucket.

> **Note**
>
> Before attempting to create an access preview for a new bucket, we recommend you call the Amazon S3 HeadBucket operation to check whether the named bucket already exists. This operation is useful to determine if a bucket exists and you have permission to access it.

Bucket policy – If the configuration is for an existing Amazon S3 bucket and you do not specify the Amazon S3 bucket policy, the access preview uses the existing policy attached to the bucket. If the access preview is for a new resource and you do not specify the Amazon S3 bucket policy, the access preview assumes a bucket without a policy. To propose deletion of an existing bucket policy, you can specify an empty string. For more information about supported bucket policy limits, see Bucket policy examples.

Bucket ACL grants – You can propose up to 100 ACL grants per bucket. If the proposed grant configuration is for an existing bucket, the access preview uses the proposed list of grant

configurations in place of the existing grants. Otherwise, the access preview uses the existing grants for the bucket.

Bucket access points – The analysis supports up to 100 access points,including multi-region access points, per bucket, including up to ten new access points you can propose per bucket. If the proposed Amazon S3 access point configuration is for an existing bucket, the access preview uses the proposed access point configuration in place of the existing access points. To propose an access point without a policy, you can provide an empty string as the access point policy. For more information about access point policy limits, see Access points restrictions and limitations.

Block public access configuration – If the proposed configuration is for an existing Amazon S3 bucket and you do not specify the configuration, the access preview uses the existing setting. If the proposed configuration is for a new bucket and you do not specify the bucket BPA configuration, the access preview uses `false`. If the proposed configuration is for a new access point or multi-region access point, and you do not specify the access point BPA configuration, the access preview uses `true`.

Preview access to your AWS KMS key

To create an access preview for a new AWS KMS key or an existing AWS KMS key that you own, you can propose a AWS KMS key configuration by specifying the key policy and the AWS KMS grant configuration.

AWS KMS key policy – If the configuration is for an existing key and you do not specify the key policy, the access preview uses the existing policy for the key. If the access preview is for a new resource and you do not specify the key policy, then the access preview uses the default key policy. The proposed key policy cannot be an empty string.

AWS KMS grants – The analysis supports up to 100 KMS grants per configuration*.* If the proposed grant configuration is for an existing key, the access preview uses the proposed list of grant configurations in place of the existing grants. Otherwise, the access preview uses the existing grants for the key.

Preview access to your IAM role

To create an access preview for a new IAM role or an existing IAM role that you own, you can propose an IAM role configuration by specifying the trust policy.

Role trust policy – If the configuration is for a new IAM role, you must specify the trust policy. If the configuration is for an existing IAM role that you own and you do not propose the trust policy,

the access preview uses the existing trust policy for the role. The proposed trust policy cannot be an empty string.

Preview access to your Amazon SQS queue

To create an access preview for a new Amazon SQS queue or an existing Amazon SQS queue that you own, you can propose an Amazon SQS queue configuration by specifying the Amazon SQS policy for the queue.

Amazon SQS queue policy – If the configuration is for an existing Amazon SQS queue and you do not specify the Amazon SQS policy, the access preview uses the existing Amazon SQS policy for the queue. If the access preview is for a new resource and you do not specify the policy, the access preview assumes an Amazon SQS queue without a policy. To propose deletion of an existing Amazon SQS queue policy, you can specify an empty string for the Amazon SQS policy.

Preview access to your Secrets Manager secret

To create an access preview for a new Secrets Manager secret or an existing Secrets Manager secret that you own, you can propose a Secrets Manager secret configuration by specifying the secret policy and optional AWS KMS encryption key.

Secret policy – If the configuration is for an existing secret and you do not specify the secret policy, the access preview uses the existing policy for the secret. If the access preview is for a new resource and you do not specify the policy, the access preview assumes a secret without a policy. To propose deletion of an existing policy, you can specify an empty string.

AWS KMS encryption key – If the proposed configuration is for a new secret and you do not specify the AWS KMS key ID, the access preview uses the default KMS key of the AWS account. If you specify an empty string for the AWS KMS key ID, the access preview uses the default KMS key of the AWS account.

Checks for validating policies

IAM Access Analyzer provides policy checks that help validate your IAM policies before you attach them to an entity. These include basic policy checks provided by policy validation to validate your policy against policy grammar and AWS best practices. You can view policy validation check findings that include security warnings, errors, general warnings, and suggestions for your policy.

You can use custom policy checks to check for new access based on your security standards. A charge is associated with each check for new access. For more details about pricing, see IAM Access Analyzer pricing.

How custom policy checks work

You can validate your policies against your specified security standards using AWS Identity and Access Management Access Analyzer custom policy checks. You can run the following types of custom policy checks:

- **Check against a reference policy**: When editing a policy, you can check whether the updated policy grants new access compared to a reference policy, such as an existing version of the policy. You can run this check when you edit a policy using the AWS Command Line Interface (AWS CLI), IAM Access Analyzer API (API), or JSON policy editor in the IAM console.

  > **Note**
  >
  > IAM Access Analyzer custom policy checks allow wildcards in the `Principal` element for reference resource policies.

- **Check against a list of IAM actions or resources**: You can check to ensure that specific IAM actions or resources are not allowed by your policy. If only actions are specified, IAM Access Analyzer checks for access of the actions on all resources in the policy. If only resources are specified, then IAM Access Analyzer checks which actions have access to the specified resources. If both actions and resources are specified, then IAM Access Analyzer checks which of the specified actions have access to the specified resources. You can run this check when you create or edit a policy using the AWS CLI or the API.
- **Check for public access**: You can check whether a resource policy can grant public access to a specified resource type. You can run this check when you create or edit a policy using the AWS CLI or the API. This type of custom policy check differs from previewing access because the check doesn't require any account or external access analyzer context. Access previews allow you to preview IAM Access Analyzer findings before deploying resource permissions, while the custom check determines whether public access might be granted by a policy.

A charge is associated with each custom policy check. For more details about pricing, see IAM Access Analyzer pricing.

You can run custom policy checks on identity and resource-based policies. Custom policy checks don't rely on pattern-matching techniques or examining access logs to determine whether new or a specified access is allowed by a policy. Similar to external access findings, custom policy checks are built on Zelkova. Zelkova translates IAM policies into equivalent logical statements, and runs a suite of general-purpose and specialized logical solvers (satisfiability modulo theories) against the problem. To check for new or specified access, IAM Access Analyzer applies Zelkova repeatedly to a policy. Queries become increasingly specific to characterize classes of behaviors that the policy allows based on the content of the policy. For more information about satisfiability modulo theories, see Satisfiability Modulo Theories.

In rare cases, IAM Access Analyzer isn't able to fully determine whether a policy statement grants new or specified access. In those cases, it errs on the side of declaring a false positive by failing the custom policy check. IAM Access Analyzer is designed to provide a comprehensive policy evaluation and strives to minimize false negatives. This approach means that IAM Access Analyzer provides a high degree of assurance that a passed check means access wasn't granted by the policy. You can inspect failed checks manually by reviewing the policy statement that's reported in the response from IAM Access Analyzer.

Examples of reference policies to check for new access

You can find examples for reference policies and learn how to set up and run a custom policy check for new access in the IAM Access Analyzer custom policy checks samples repository on GitHub.

Before using these examples

Before you use these sample reference policies, do the following:

- Carefully review and customize the reference policies for your unique requirements.
- Thoroughly test the reference policies in your environment with the AWS services that you use.

  The reference policies demonstrate the implementation and use of custom policy checks. They're ***not*** intended to be interpreted as official AWS recommendations or best practices to be implemented exactly as shown. It is your responsibility to carefully test reference policies for their suitability to solve the security requirements for your environment.
- Custom policy checks are environment-agnostic in their analysis. Their analysis only considers information contained within the input policies. For example, custom policy checks can't check whether an account is a member of a specific AWS organization.

Therefore, the custom policy checks can't compare new access based on condition key values for the `aws:PrincipalOrgId` and `aws:PrincipalAccount` condition keys.

Inspect failed custom policy checks

When a custom policy check fails, the response from IAM Access Analyzer includes the statement ID (`Sid`) of the policy statement that caused the check to fail. Although the statement ID is an optional policy element, we recommend that you add a statement ID for every policy statement. The custom policy check also returns a statement index to help identify the reason for the check failure. The statement index follows zero-based numbering, where the first statement is referenced as 0. When there are multiple statements that cause a check to fail, the check returns only one statement ID at a time. We recommend that you fix the statement highlighted in the reason and rerun the check until it passes.

Validate policies with IAM Access Analyzer

You can validate your policies using AWS Identity and Access Management Access Analyzer policy validation. You can create or edit a policy using the AWS CLI, AWS API, or JSON policy editor in the IAM console. IAM Access Analyzer validates your policy against IAM policy grammar and AWS best practices. You can view policy validation check findings that include security warnings, errors, general warnings, and suggestions for your policy. These findings provide actionable recommendations that help you author policies that are functional and conform to security best practices. To view a list of the basic policy checks that are run by IAM Access Analyzer, see IAM policy validation check reference.

Validating policies in IAM (console)

You can view findings generated by IAM Access Analyzer policy validation when you create or edit a managed policy in the IAM console. You can also view these findings for inline user or role policies. IAM Access Analyzer does not generate these findings for inline group policies.

To view findings generated by policy checks for IAM JSON policies

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Begin creating or editing a policy using one of the following methods:

a. To create a new managed policy, go to the **Policies** page and create a new policy. For more information, see Creating policies using the JSON editor.

b. To view policy checks for an existing customer managed policy, go the **Policies** page, choose the name of a policy, and then choose **Edit**. For more information, see Editing customer managed policies (console).

c. To view policy checks for an inline policy on a user or role, go the **Users** or **Roles** page, choose the name of a user or role, choose the name of the policy on the **Permissions** tab and then choose **Edit**. For more information, see Editing inline policies (console).

3. In the policy editor, choose the **JSON** tab.

4. In the policy validation pane below the policy, choose one or more of the following tabs. The tab names also indicate the number of each finding type for your policy.

   - **Security** – View warnings if your policy allows access that AWS considers a security risk because the access is overly permissive.
   - **Errors** – View errors if your policy includes lines that prevent the policy from functioning.
   - **Warnings** – View warnings if your policy doesn't conform to best practices, but the issues are not security risks.
   - **Suggestions** – View suggestions if AWS recommends improvements that don't impact the permissions of the policy.

5. Review the finding details provided by the IAM Access Analyzer policy check. Each finding indicates the location of the reported issue. To learn more about what causes the issue and how to resolve it, choose the **Learn more** link next to the finding. You can also search for the policy check associated with each finding in the Access Analyzer policy checks reference page.

6. Optional. If you are editing an existing policy, you can run a custom policy check to determine whether your updated policy grants new access compared to the existing version. In the policy validation pane below the policy, choose the **Check for new access** tab and then choose **Check policy**. If the modified permissions grant new access, the statement will be highlighted in the policy validation pane. If you do not intend to grant new access, update the policy statement and choose **Check policy** until no new access is detected. For more information, see Validate policies with IAM Access Analyzer custom policy checks.

> **Note**
>
> A charge is associated with each check for new access. For more details on pricing, see IAM Access Analyzer pricing.

7. Update your policy to resolve the findings.

> **Important**
>
> Test new or edited policies thoroughly before implementing them in your production workflow.

8. When you are finished, choose **Next**. The Policy validator reports any syntax errors that are not reported by IAM Access Analyzer.

> **Note**
>
> You can switch between the **Visual** and **JSON** tabs anytime. However, if you make changes or choose **Next** in the **Visual** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

9. For new policies, on the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review the **Permissions defined in this policy** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

   For existing policies, on the **Review and save** page, review the **Permissions defined in this policy** to see the permissions that are granted by your policy. Choose the **Set this new version as the default.** checkbox to save the updated version as the default version of the policy. Then choose **Save changes** to save your work.

Validating policies using IAM Access Analyzer (AWS CLI or AWS API)

You can view findings generated by IAM Access Analyzer policy validation from the AWS Command Line Interface (AWS CLI).

To view findings generated by IAM Access Analyzer policy validation (AWS CLI or AWS API)

Use one of the following:

- AWS CLI: aws accessanalyzer validate-policy
- AWS API: ValidatePolicy

IAM policy validation check reference

You can validate your policies using AWS Identity and Access Management Access Analyzer policy validation. You can create or edit a policy using the AWS CLI, AWS API, or JSON policy editor in the IAM console. IAM Access Analyzer validates your policy against IAM policy grammar and AWS best practices. You can view policy validation check findings that include security warnings, errors, general warnings, and suggestions for your policy. These findings provide actionable recommendations that help you author policies that are functional and conform to security best practices. The list of basic policy checks provided by IAM Access Analyzer are shared below. There is no additional charge associated with running the policy validation checks. To learn more about validating policies using policy validation, see Validate policies with IAM Access Analyzer.

Error – ARN account not allowed

Issue code: ARN_ACCOUNT_NOT_ALLOWED

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
ARN account not allowed: The service {{service}} does not support specifying an account
 ID in the resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} does not support specifying an account ID in
 the resource ARN."
```

Resolving the error

Remove the account ID from the resource ARN. The resource ARNs for some AWS services do not support specifying an account ID.

For example, Amazon S3 does not support an account ID as a namespace in bucket ARNs. An Amazon S3 bucket name is globally unique, and the namespace is shared by all AWS accounts. To view all of the resource types available in Amazon S3, see Resource types defined by Amazon S3 in the *Service Authorization Reference*.

Related terms

- Policy resources
- Account Identifiers
- Resource ARNs
- AWS service resources with ARN formats

Error – ARN Region not allowed

Issue code: ARN_REGION_NOT_ALLOWED

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
ARN Region not allowed: The service {{service}} does not support specifying a Region in
 the resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} does not support specifying a Region in the
 resource ARN."
```

Resolving the error

Remove the Region from the resource ARN. The resource ARNs for some AWS services do not support specifying a Region.

For example, IAM is a global service. The Region portion of an IAM resource ARN is always kept blank. IAM resources are global, like an AWS account is today. For example, after you sign in as an IAM user, you can access AWS services in any geographic region.

- Policy resources
- Resource ARNs
- AWS service resources with ARN formats

Error – Data type mismatch

Issue code: DATA_TYPE_MISMATCH

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Data type mismatch: The text does not match the expected JSON data type {{data_type}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The text does not match the expected JSON data type {{data_type}}."
```

Resolving the error

Update the text to use the supported data type.

For example, the `Version` global condition key requires a `String` data type. If you provide a date or an integer, the data type won't match.

Related terms

- Global condition keys
- IAM JSON policy elements: Condition operators

Error – Duplicate keys with different case

Issue code: DUPLICATE_KEYS_WITH_DIFFERENT_CASE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Duplicate keys with different case: The condition key {{key}} appears more than once
 with different capitalization in the same condition block. Remove the duplicate
 condition keys.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{key}} appears more than once with different
 capitalization in the same condition block. Remove the duplicate condition keys."
```

Resolving the error

Review the similar condition keys within the same condition block and use the same capitalization for all instances.

A *condition block* is the text within the `Condition` element of a policy statement. Condition key *names* are not case-sensitive. The case-sensitivity of condition key *values* depends on the condition operator that you use. For more information about case-sensitivity in condition keys, see IAM JSON policy elements: Condition.

Related terms

- Conditions
- Condition block
- Global condition keys
- AWS service condition keys

Error – Invalid action

Issue code: INVALID_ACTION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid action: The action {{action}} does not exist. Did you mean {{valid_action}}?
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The action {{action}} does not exist. Did you mean
 {{valid_action}}?"
```

Resolving the error

The action that you specified is not valid. This can happen if you mis-type the service prefix or the action name, or if the action is not yet recognized by IAM policy validation. For some common issues, the policy check returns a suggested action.

Related terms

- Policy actions
- AWS service actions

AWS managed policies with this error

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

AWS managed policies can sometimes include invalid actions in their policy statements. Invalid actions do not affect the permissions granted by the policy. When using an AWS managed policy as a reference to create your managed policy, AWS recommends that you review the invalid actions to fix any errors.

Error – Invalid ARN account

Issue code: INVALID_ARN_ACCOUNT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN account: The resource ARN account ID {{account}} is not valid. Provide a
 12-digit account ID.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource ARN account ID {{account}} is not valid. Provide a 12-
digit account ID."
```

Resolving the error

Update the account ID in the resource ARN. Account IDs are 12-digit integers. To learn how to view your account ID, see Finding your AWS account ID.

Related terms

- Policy resources
- Account Identifiers
- Resource ARNs
- AWS service resources with ARN formats

Error – Invalid ARN prefix

Issue code: INVALID_ARN_PREFIX

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN prefix: Add the required prefix (arn) to the resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add the required prefix (arn) to the resource ARN."
```

Resolving the error

AWS resource ARNs must include the required `arn:` prefix.

Related terms

- Policy resources
- Resource ARNs
- AWS service resources with ARN formats

Error – Invalid ARN Region

Issue code: INVALID_ARN_REGION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN Region: The Region {{region}} is not valid for this resource. Update the
 resource ARN to include a supported Region.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Region {{region}} is not valid for this resource. Update the
 resource ARN to include a supported Region."
```

Resolving the error

The resource type is not supported in the specified Region. For a table of AWS services supported in each Region, see the Region table.

Related terms

- Policy resources
- Resource ARNs
- Region names and codes

Error – Invalid ARN resource

Issue code: INVALID_ARN_RESOURCE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN resource: Resource ARN does not match the expected ARN format. Update the
 resource portion of the ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Resource ARN does not match the expected ARN format. Update the
 resource portion of the ARN."
```

Resolving the error

The resource ARN must match the specifications for known resource types. To view the expected ARN format for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service to view its resource types and ARN formats.

Related terms

- Policy resources
- Resource ARNs
- AWS service resources with ARN formats

Error – Invalid ARN service case

Issue code: INVALID_ARN_SERVICE_CASE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid ARN service case: Update the service name {{service}} in the resource ARN to
 use all lowercase letters.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Update the service name {{service}} in the resource ARN to use all
 lowercase letters."
```

Resolving the error

The service in the resource ARN must match the specifications (including capitalization) for service prefixes. To view the prefix for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service and locate its prefix in the first sentence.

Related terms

- Policy resources
- Resource ARNs
- AWS service resources with ARN formats

Error – Invalid condition data type

Issue code: INVALID_CONDITION_DATA_TYPE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid condition data type: The condition value data types do not match. Use condition
 values of the same JSON data type.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition value data types do not match. Use condition values of
 the same JSON data type."
```

Resolving the error

The value in the condition key-value pair must match the data type of the condition key and condition operator. To view the condition key data type for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service to view the condition keys for that service.

For example, the `CurrentTime` global condition key supports the `Date` condition operator. If you provide a string or an integer for the value in the condition block, the data type won't match.

Related terms

- Conditions
- Condition block
- IAM JSON policy elements: Condition operators
- Global condition keys
- AWS service condition keys

Error – Invalid condition key format

Issue code: INVALID_CONDITION_KEY_FORMAT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid condition key format: The condition key format is not valid. Use the format
 service:keyname.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key format is not valid. Use the format
 service:keyname."
```

Resolving the error

The key in the condition key-value pair must match the specifications for the service. To view the condition keys for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service to view the condition keys for that service.

Related terms

- Conditions
- Global condition keys
- AWS service condition keys

Error – Invalid condition multiple Boolean

Issue code: INVALID_CONDITION_MULTIPLE_BOOLEAN

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid condition multiple Boolean: The condition key does not support multiple Boolean
 values. Use a single Boolean value.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key does not support multiple Boolean values. Use a
 single Boolean value."
```

Resolving the error

The key in the condition key-value pair expects a single Boolean value. When you provide multiple Boolean values, the condition match might not return the results that you expect.

To view the condition keys for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service to view the condition keys for that service.

- Conditions
- Global condition keys

- AWS service condition keys

Error – Invalid condition operator

Issue code: INVALID_CONDITION_OPERATOR

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid condition operator: The condition operator {{operator}} is not valid. Use a
 valid condition operator.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition operator {{operator}} is not valid. Use a valid
 condition operator."
```

Resolving the error

Update the condition to use a supported condition operator.

Related terms

- IAM JSON policy elements: Condition operators
- Condition element
- Overview of JSON policies

Error – Invalid effect

Issue code: INVALID_EFFECT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid effect: The effect {{effect}} is not valid. Use Allow or Deny.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The effect {{effect}} is not valid. Use Allow or Deny."
```

Resolving the error

Update the `Effect` element to use a valid effect. Valid values for `Effect` are **`Allow`** and **`Deny`**.

Related terms

- Effect element
- Overview of JSON policies

Error – Invalid global condition key

Issue code: INVALID_GLOBAL_CONDITION_KEY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid global condition key: The condition key {{key}} does not exist. Use a valid
 condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{key}} does not exist. Use a valid condition
 key."
```

Resolving the error

Update the condition key in the condition key-value pair to use a supported global condition key.

Global condition keys are condition keys with an `aws:` prefix. AWS services can support global condition keys or provide service-specific keys that include their service prefix. For example, IAM condition keys include the `iam:` prefix. For more information, see Actions, Resources, and Condition Keys for AWS Services and choose the service whose keys you want to view.

Related terms

- Global condition keys

Error – Invalid partition

Issue code: INVALID_PARTITION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid partition: The resource ARN for the service {{service}} does not support the
 partition {{partition}}. Use the supported values: {{partitions}}
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource ARN for the service {{service}} does not support the
 partition {{partition}}. Use the supported values: {{partitions}}"
```

Resolving the error

Update the resource ARN to include a supported partition. If you included a supported partition, then the service or resource might not support the partition that you included.

A *partition* is a group of AWS Regions. Each AWS account is scoped to one partition. In Classic Regions, use the `aws` partition. In China Regions, use `aws-cn`.

Related terms

- Amazon Resource Names (ARNs) - Partitions

Error – Invalid policy element

Issue code: INVALID_POLICY_ELEMENT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid policy element: The policy element {{element}} is not valid.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy element {{element}} is not valid."
```

Resolving the error

Update the policy to include only supported JSON policy elements.

Related terms

- JSON policy elements

Error – Invalid principal format

Issue code: INVALID_PRINCIPAL_FORMAT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid principal format: The Principal element contents are not valid. Specify a key-
value pair in the Principal element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Principal element contents are not valid. Specify a key-value
 pair in the Principal element."
```

Resolving the error

Update the principal to use a supported key-value pair format.

You can specify a principal in a resource-based policy, but not an identity-based policy.

For example, to define access for everyone in an AWS account, use the following principal in your policy:

```
"Principal": { "AWS": "123456789012" }
```

Related terms

- JSON policy elements: Principal
- Identity-based policies and resource-based policies

Error – Invalid principal key

Issue code: INVALID_PRINCIPAL_KEY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid principal key: The principal key {{principal-key}} is not valid.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The principal key {{principal-key}} is not valid."
```

Resolving the error

Update the key in the principal key-value pair to use a supported principal key. The following are supported principal keys:

- AWS
- CanonicalUser
- Federated
- Service

Related terms

- Principal element

Error – Invalid Region

Issue code: INVALID_REGION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid Region: The Region {{region}} is not valid. Update the condition value to a
 supported Region.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Region {{region}} is not valid. Update the condition value to a
 supported Region."
```

Resolving the error

Update the value of the condition key-value pair to include a supported Region. For a table of AWS services supported in each Region, see the Region table.

Related terms

- Policy resources
- Resource ARNs
- Region names and codes

Error – Invalid service

Issue code: INVALID_SERVICE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid service: The service {{service}} does not exist. Use a valid service name.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} does not exist. Use a valid service name."
```

Resolving the error

The service prefix in the action or condition key must match the specifications (including capitalization) for service prefixes. To view the prefix for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service and locate its prefix in the first sentence.

Related terms

- Known services and their actions, resources, and condition keys

Error – Invalid service condition key

Issue code: INVALID_SERVICE_CONDITION_KEY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid service condition key: The condition key {{key}} does not exist in the service
 {{service}}. Use a valid condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{key}} does not exist in the service {{service}}.
 Use a valid condition key."
```

Resolving the error

Update the key in the condition key-value pair to use a known condition key for the service. Global condition key names begin with the `aws` prefix. AWS services can provide service-specific keys that include their service prefix. To view the prefix for a service, see Actions, resources, and condition keys for AWS services.

Related terms

- Global condition keys
- Known services and their actions, resources, and condition keys

Error – Invalid service in action

Issue code: INVALID_SERVICE_IN_ACTION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid service in action: The service {{service}} specified in the action does not
 exist. Did you mean {{service2}}?
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service {{service}} specified in the action does not exist. Did
 you mean {{service2}}?"
```

Resolving the error

The service prefix in the action must match the specifications (including capitalization) for service prefixes. To view the prefix for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service and locate its prefix in the first sentence.

Related terms

- Action element
- Known services and their actions

Error – Invalid variable for operator

Issue code: INVALID_VARIABLE_FOR_OPERATOR

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid variable for operator: Policy variables can only be used with String and ARN
 operators.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Policy variables can only be used with String and ARN operators."
```

Resolving the error

You can use policy variables in the `Resource` element and in string comparisons in the `Condition` element. Conditions support variables when you use string operators or ARN operators. String operators include `StringEquals`, `StringLike`, and `StringNotLike`. ARN operators include `ArnEquals` and `ArnLike`. You can't use a policy variable with other operators, such as Numeric, Date, Boolean, Binary, IP Address, or Null operators.

Related terms

- Using policy variables in the Condition element
- Condition element

Error – Invalid version

Issue code: INVALID_VERSION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid version: The version {{version}} is not valid. Use one of the following
 versions: {{versions}}
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The version {{version}} is not valid. Use one of the following
 versions: {{versions}}"
```

Resolving the error

The `Version` policy element specifies the language syntax rules that AWS uses to process a policy. To use all of the available policy features, include the latest `Version` element before the `Statement` element in all of your policies.

```
"Version": "2012-10-17"
```

Related terms

- Version element

Error – Json syntax error

Issue code: JSON_SYNTAX_ERROR

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Json syntax error: Fix the JSON syntax error at index {{index}} line {{line}} column
 {{column}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Fix the JSON syntax error at index {{index}} line {{line}} column
 {{column}}."
```

Resolving the error

Your policy includes a syntax error. Check your JSON syntax.

Related terms

- JSON validator
- IAM JSON policy elements reference
- Overview of JSON policies

Error – Json syntax error

Issue code: JSON_SYNTAX_ERROR

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Json syntax error: Fix the JSON syntax error.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Fix the JSON syntax error."
```

Resolving the error

Your policy includes a syntax error. Check your JSON syntax.

Related terms

- JSON validator
- IAM JSON policy elements reference
- Overview of JSON policies

Error – Missing action

Issue code: MISSING_ACTION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing action: Add an Action or NotAction element to the policy statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add an Action or NotAction element to the policy statement."
```

Resolving the error

AWS JSON policies must include an `Action` or `NotAction` element.

Related terms

- Action element
- NotAction element
- Overview of JSON policies

Error – Missing ARN field

Issue code: MISSING_ARN_FIELD

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing ARN field: Resource ARNs must include at least {{fields}} fields in the
 following structure: arn:partition:service:region:account:resource
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Resource ARNs must include at least {{fields}} fields in the
 following structure: arn:partition:service:region:account:resource"
```

Resolving the error

All of the fields in the resource ARN must match the specifications for a known resource type. To view the expected ARN format for a service, see Actions, resources, and condition keys for AWS services. Choose the name of the service to view its resource types and ARN formats.

Related terms

- Policy resources
- Resource ARNs
- AWS service resources with ARN formats

Error – Missing ARN Region

Issue code: MISSING_ARN_REGION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing ARN Region: Add a Region to the {{service}} resource ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a Region to the {{service}} resource ARN."
```

Resolving the error

The resource ARNs for most AWS services require that you specify a Region. For a table of AWS services supported in each Region, see the Region table.

Related terms

- Policy resources
- Resource ARNs
- Region names and codes

Error – Missing effect

Issue code: MISSING_EFFECT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing effect: Add an Effect element to the policy statement with a value of Allow or
 Deny.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add an Effect element to the policy statement with a value of Allow
 or Deny."
```

Resolving the error

AWS JSON policies must include an `Effect` element with a value of **`Allow`** and **`Deny`**.

Related terms

- Effect element

- Overview of JSON policies

Error – Missing principal

Issue code: MISSING_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing principal: Add a Principal element to the policy statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a Principal element to the policy statement."
```

Resolving the error

Resource-based policies must include a `Principal` element.

For example, to define access for everyone in an AWS account, use the following principal in your policy:

```
"Principal": { "AWS": "123456789012" }
```

Related terms

- Principal element
- Identity-based policies and resource-based policies

Error – Missing qualifier

Issue code: MISSING_QUALIFIER

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing qualifier: The request context key {{key}} has multiple values. Use the
 ForAllValues or ForAnyValue condition key qualifiers in your policy.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The request context key {{key}} has multiple values. Use the
 ForAllValues or ForAnyValue condition key qualifiers in your policy."
```

Resolving the error

In the `Condition` element, you build expressions in which you use condition operators like equal or less than to compare a condition in the policy against keys and values in the request context. For requests that include multiple values for a single condition key, you must enclose the conditions within brackets like an array ("Key2":["Value2A", "Value2B"]). You must also use the `ForAllValues` or `ForAnyValue` set operators with the `StringLike` condition operator. These qualifiers add set-operation functionality to the condition operator so that you can test multiple request values against multiple condition values.

Related terms

- Multivalued context keys
- Condition element

AWS managed policies with this error

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

The following AWS managed policies include a missing qualifier for condition keys in their policy statements. When using the AWS managed policy as a reference to create your customer managed policy, AWS recommends that you add the `ForAllValues` or `ForAnyValue` condition key qualifiers to your `Condition` element.

- AWSGlueConsoleSageMakerNotebookFullAccess

Error – Missing resource

Issue code: MISSING_RESOURCE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing resource: Add a Resource or NotResource element to the policy statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a Resource or NotResource element to the policy statement."
```

Resolving the error

All policies except role trust policies must include a `Resource` or `NotResource` element.

Related terms

- Resource element
- NotResource element
- Identity-based policies and resource-based policies
- Overview of JSON policies

Error – Missing statement

Issue code: MISSING_STATEMENT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing statement: Add a statement to the policy
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a statement to the policy"
```

Resolving the error

A JSON policy must include a statement.

Related terms

- JSON policy elements

Error – Null with if exists

Issue code: NULL_WITH_IF_EXISTS

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Null with if exists: The Null condition operator cannot be used with the IfExists
 suffix. Update the operator or the suffix.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Null condition operator cannot be used with the IfExists suffix.
 Update the operator or the suffix."
```

Resolving the error

You can add `IfExists` to the end of any condition operator name except the `Null` condition operator. Use a `Null` condition operator to check if a condition key is present at the time of authorization. Use `...ifExists` to say "If the policy key is present in the context of the request, process the key as specified in the policy. If the key is not present, evaluate the condition element as true."

Related terms

- ...IfExists condition operators
- Null condition operator
- Condition element

Error – SCP syntax error action wildcard

Issue code: SCP_SYNTAX_ERROR_ACTION_WILDCARD

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
SCP syntax error action wildcard: SCP actions can include wildcards (*) only at the end
 of a string. Update {{action}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "SCP actions can include wildcards (*) only at the end of a string.
 Update {{action}}."
```

Resolving the error

AWS Organizations service control policies (SCPs) support specifying values in the `Action` or `NotAction` elements. However, these values can include wildcards (*) only at the end of the string. This means that you can specify `iam:Get*` but not `iam:*role`.

To specify multiple actions, AWS recommends that you list them individually.

Related terms

- SCP Action and NotAction elements
- SCP evaluation
- AWS Organizations service control policies
- IAM JSON policy elements: Action

Error – SCP syntax error principal

Issue code: SCP_SYNTAX_ERROR_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
SCP syntax error principal: SCPs do not support specifying principals. Remove the
 Principal or NotPrincipal element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "SCPs do not support specifying principals. Remove the Principal or
 NotPrincipal element."
```

Resolving the error

AWS Organizations service control policies (SCPs) do not support the `Principal` or `NotPrincipal` elements.

You can specify the Amazon Resource Name (ARN) using the `aws:PrincipalArn` global condition key in the `Condition` element.

Related terms

- SCP syntax
- Global condition keys for principals

Error – Unique Sids required

Issue code: UNIQUE_SIDS_REQUIRED

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unique Sids required: Duplicate statement IDs are not supported for this policy type.
 Update the Sid value.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Duplicate statement IDs are not supported for this policy type.
 Update the Sid value."
```

Resolving the error

For some policy types, statement IDs must be unique. The `Sid` (statement ID) element allows you to enter an optional identifier that you provide for the policy statement. You can assign a statement ID value to each statement in a statement array using the `SID` element. In services that let you specify an ID element, such as SQS and SNS, the `Sid` value is just a sub-ID of the policy document's ID. For example, in IAM, the `Sid` value must be unique within a JSON policy.

Related terms

- IAM JSON policy elements: Sid

Error – Unsupported action in policy

Issue code: UNSUPPORTED_ACTION_IN_POLICY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported action in policy: The action {{action}} is not supported for the resource-
based policy attached to the resource type {{resourceType}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The action {{action}} is not supported for the resource-based policy
 attached to the resource type {{resourceType}}."
```

Resolving the error

Some actions aren't supported in the `Action` element in the resource-based policy attached to a different resource type. For example, AWS Key Management Service actions aren't supported in Amazon S3 bucket policies. Specify an action that is supported by resource type attached to your resource-based policy.

Related terms

- JSON policy elements: Action

Error – Unsupported element combination

Issue code: UNSUPPORTED_ELEMENT_COMBINATION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported element combination: The policy elements {{element1}} and {{element2}} can
 not be used in the same statement. Remove one of these elements.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy elements {{element1}} and {{element2}} can not be used in
 the same statement. Remove one of these elements."
```

Resolving the error

Some combinations of JSON policy elements can't be used together. For example, you cannot use both `Action` and `NotAction` in the same policy statement. Other pairs that are mutually exclusive include `Principal/NotPrincipal` and `Resource/NotResource`.

Related terms

- IAM JSON policy elements reference
- Overview of JSON policies

Error – Unsupported global condition key

Issue code: UNSUPPORTED_GLOBAL_CONDITION_KEY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported global condition key: The condition key aws:ARN is not supported. Use
 aws:PrincipalArn or aws:SourceArn instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key aws:ARN is not supported. Use aws:PrincipalArn or
 aws:SourceArn instead."
```

Resolving the error

AWS does not support using the specified global condition key. Depending on your use case, you can use the `aws:PrincipalArn` or `aws:SourceArn` global condition keys. For example, instead of `aws:ARN`, use the `aws:PrincipalArn` to compare the Amazon Resource Name (ARN) of the principal that made the request with the ARN that you specify in the policy. Alternatively, use the `aws:SourceArn` global condition key to compare the Amazon Resource Name (ARN) of the resource making a service-to-service request with the ARN that you specify in the policy.

Related terms

- AWS global condition context keys

Error – Unsupported principal

Issue code: UNSUPPORTED_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported principal: The policy type {{policy_type}} does not support the Principal
 element. Remove the Principal element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy type {{policy_type}} does not support the Principal
 element. Remove the Principal element."
```

Resolving the error

The `Principal` element specifies the principal that is allowed or denied access to a resource. You cannot use the `Principal` element in an IAM identity-based policy. You can use it in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in a resource. For example, you can embed policies in an Amazon S3 bucket or an AWS KMS key.

Related terms

- AWS JSON policy elements: Principal
- Cross account resource access in IAM

Error – Unsupported resource ARN in policy

Issue code: UNSUPPORTED_RESOURCE_ARN_IN_POLICY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported resource ARN in policy: The resource ARN is not supported for the resource-
based policy attached to the resource type {{resourceType}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource ARN is not supported for the resource-based policy
 attached to the resource type {{resourceType}}."
```

Resolving the error

Some resource ARNs aren't supported in the `Resource` element of the resource-based policy when the policy is attached to a different resource type. For example, AWS KMS ARNs aren't supported in the `Resource` element for Amazon S3 bucket policies. Specify a resource ARN that is supported by a resource type attached to your resource-based policy.

Related terms

- JSON policy elements: Action

Error – Unsupported Sid

Issue code: UNSUPPORTED_SID

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported Sid: Update the characters in the Sid element to use one of the following
 character types: [a-z, A-Z, 0-9]
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Update the characters in the Sid element to use one of the following
 character types: [a-z, A-Z, 0-9]"
```

Resolving the error

The `Sid` element supports uppercase letters, lowercase letters, and numbers.

Related terms

- IAM JSON policy elements: Sid

Error – Unsupported wildcard in principal

Issue code: UNSUPPORTED_WILDCARD_IN_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported wildcard in principal: Wildcards (*, ?) are not supported with the
 principal key {{principal_key}}. Replace the wildcard with a valid principal value.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Wildcards (*, ?) are not supported with the principal key
 {{principal_key}}. Replace the wildcard with a valid principal value."
```

Resolving the error

The `Principal` element structure supports using a key-value pair. The principal value specified in the policy includes a wildcard (*). You can't include a wildcard with the principal key that you specified. For example, when you specify users in a `Principal` element, you cannot use a wildcard to mean "all users". You must name a specific user or users. Similarly, when you specify an assumed-role session, you cannot use a wildcard to mean "all sessions". You must name a specific session. You also cannot use a wildcard to match part of a name or an ARN.

To resolve this finding, remove the wildcard and provide a more specific principal.

Related terms

- AWS JSON policy elements: Principal

Error – Missing brace in variable

Issue code: MISSING_BRACE_IN_VARIABLE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing brace in variable: The policy variable is missing a closing curly brace. Add }
 after the variable text.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy variable is missing a closing curly brace. Add } after
 the variable text."
```

Resolving the error

Policy variable structure supports using a $ prefix followed by a pair of curly braces ({ }). Inside the ${ } characters, include the name of the value from the request that you want to use in the policy.

To resolve this finding, add the missing brace to make sure the full opening and closing set of braces is present.

Related terms

- IAM policy elements: Variables

Error – Missing quote in variable

Issue code: MISSING_QUOTE_IN_VARIABLE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing quote in variable: The policy variable default value must begin and end with a
 single quote. Add the missing quote.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy variable default value must begin and end with a single
 quote. Add the missing quote."
```

Resolving the error

When you add a variable to your policy, you can specify a default value for the variable. If a variable is not present, AWS uses the default text that you provide.

To add a default value to a variable, surround the default value with single quotes (`' '`), and separate the variable text and the default value with a comma and space (`, `).

For example, if a principal is tagged with `team=yellow`, they can access the `amzn-s3-demo-bucket` Amazon S3 bucket with the name `amzn-s3-demo-bucket-yellow`. A policy with this resource might allow team members to access their own resources, but not those of other teams. For users without team tags, you might set a default value of `company-wide`. These users can access only the `amzn-s3-demo-bucket-company-wide` bucket where they can view broad information, such as instructions for joining a team.

```
"Resource":"arn:aws:s3:::amzn-s3-demo-bucket-${aws:PrincipalTag/team, 'company-wide'}"
```

Related terms

- IAM policy elements: Variables

Error – Unsupported space in variable

Issue code: UNSUPPORTED_SPACE_IN_VARIABLE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported space in variable: A space is not supported within the policy variable
 text. Remove the space.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "A space is not supported within the policy variable text. Remove the
 space."
```

Resolving the error

Policy variable structure supports using a $ prefix followed by a pair of curly braces ({ }). Inside the ${ } characters, include the name of the value from the request that you want to use in the policy. Although you can include a space when you specify a default variable, you cannot include a space in the variable name.

Related terms

- IAM policy elements: Variables

Error – Empty variable

Issue code: EMPTY_VARIABLE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty variable: Empty policy variable. Remove the ${ } variable structure or provide a
 variable within the structure.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Empty policy variable. Remove the ${ } variable structure or provide
 a variable within the structure."
```

Resolving the error

Policy variable structure supports using a $ prefix followed by a pair of curly braces ({ }). Inside the ${ } characters, include the name of the value from the request that you want to use in the policy.

Related terms

- IAM policy elements: Variables

Error – Variable unsupported in element

Issue code: VARIABLE_UNSUPPORTED_IN_ELEMENT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Variable unsupported in element: Policy variables are supported in the Resource and
 Condition elements. Remove the policy variable {{variable}} from this element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Policy variables are supported in the Resource and Condition
 elements. Remove the policy variable {{variable}} from this element."
```

Resolving the error

You can use policy variables in the `Resource` element and in string comparisons in the `Condition` element.

Related terms

- IAM policy elements: Variables

Error – Variable unsupported in version

Issue code: VARIABLE_UNSUPPORTED_IN_VERSION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Variable unsupported in version: To include variables in your policy, use the policy
 version 2012-10-17 or later.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "To include variables in your policy, use the policy version
 2012-10-17 or later."
```

Resolving the error

To use policy variables, you must include the `Version` element and set it to a version that supports policy variables. Variables were introduced in version `2012-10-17`. Earlier versions of the policy language don't support policy variables. If you don't set the `Version` to `2012-10-17` or later, variables like `${aws:username}` are treated as literal strings in the policy.

A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, is created when you change a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy.

Related terms

- IAM policy elements: Variables
- IAM JSON policy elements: Version

Error – Private IP address

Issue code: PRIVATE_IP_ADDRESS

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Private IP address: aws:SourceIp works only for public IP address ranges. The values
 for condition key aws:SourceIp include only private IP addresses and will not have the
 desired effect. Update the value to include only public IP addresses.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "aws:SourceIp works only for public IP address ranges. The values
 for condition key aws:SourceIp include only private IP addresses and will not have the
 desired effect. Update the value to include only public IP addresses."
```

Resolving the error

The global condition key `aws:SourceIp` works only for public IP address ranges. You receive this error when your policy allows only private IP addresses. In this case, the condition would never match.

- aws:SourceIp global condition key
- IAM JSON policy elements: Condition

Error – Private NotIpAddress

Issue code: PRIVATE_NOT_IP_ADDRESS

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Private NotIpAddress: The values for condition key aws:SourceIp include only private
 IP addresses and has no effect. aws:SourceIp works only for public IP address ranges.
 Update the value to include only public IP addresses.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The values for condition key aws:SourceIp include only private IP
 addresses and has no effect. aws:SourceIp works only for public IP address ranges.
 Update the value to include only public IP addresses."
```

Resolving the error

The global condition key `aws:SourceIp` works only for public IP address ranges. You receive this error when you use the `NotIpAddress` condition operator and list only private IP addresses. In this case, the condition would always match and would be ineffective.

- aws:SourceIp global condition key
- IAM JSON policy elements: Condition

Error – Policy size exceeds SCP quota

Issue code: POLICY_SIZE_EXCEEDS_SCP_QUOTA

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Policy size exceeds SCP quota: The {{policySize}} characters in the service control
 policy (SCP) exceed the {{policySizeQuota}} character maximum for SCPs. We recommend
 that you use multiple granular policies.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{policySize}} characters in the service control policy (SCP)
 exceed the {{policySizeQuota}} character maximum for SCPs. We recommend that you use
 multiple granular policies."
```

Resolving the error

AWS Organizations service control policies (SCPs) support specifying values in the `Action` or `NotAction` elements. However, these values can include wildcards (*) only at the end of the string. This means that you can specify `iam:Get*` but not `iam:*role`.

To specify multiple actions, AWS recommends that you list them individually.

Related terms

- Quotas for AWS Organizations

- AWS Organizations service control policies

Error – Invalid service principal format

Issue code: INVALID_SERVICE_PRINCIPAL_FORMAT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid service principal format: The service principal does not match the expected
 format. Use the format {{expectedFormat}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The service principal does not match the expected format. Use the
 format {{expectedFormat}}."
```

Resolving the error

The value in the condition key-value pair must match a defined service principal format.

A *service principal* is an identifier that is used to grant permissions to a service. You can specify a service principal in the `Principal` element or as a value for some global condition keys and service-specific keys. The service principal is defined by each service.

The identifier for a service principal includes the service name, and is usually in the following format in all lowercase letters:

`service-name``.amazonaws.com`

Some service-specific keys may use a different format for service principals. For example, the `kms:ViaService` condition key requires the following format for service principals in all lowercase letters:

`service-name.AWS_region``.amazonaws.com`

Related terms

- Service principals
- AWS global condition keys
- `kms:ViaService` condition key

Error – Missing tag key in condition

Issue code: MISSING_TAG_KEY_IN_CONDITION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing tag key in condition: The condition key {{conditionKeyName}} must include a tag
 key to control access based on tags. Use the format {{conditionKeyName}}tag-key and
 specify a key name for tag-key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{conditionKeyName}} must include a tag key to
 control access based on tags. Use the format {{conditionKeyName}}tag-key and specify a
 key name for tag-key."
```

Resolving the error

To control access based on tags, you provide tag information in the condition element of a policy.

For example, to control access to AWS resources, you include the `aws:ResourceTag` condition key. This key requires the format `aws:ResourceTag/`*`tag-key`*. To specify the tag key `owner` and the tag value `JaneDoe` in a condition, use the following format.

```
"Condition": {
    "StringEquals": {"aws:ResourceTag/owner": "JaneDoe"}
}
```

Related terms

- Controlling access using tags
- Conditions
- Global condition keys
- AWS service condition keys

Error – Invalid vpc format

Issue code: INVALID_VPC_FORMAT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid vpc format: The VPC identifier in the condition key value is not valid. Use the
 prefix 'vpc-' followed by 8 or 17 alphanumeric characters.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The VPC identifier in the condition key value is not valid. Use the
 prefix 'vpc-' followed by 8 or 17 alphanumeric characters."
```

Resolving the error

The `aws:SourceVpc` condition key must use the prefix `vpc-` followed by either 8 or 17 alphanumeric characters, for example, `vpc-11223344556677889` or `vpc-12345678`.

Related terms

- AWS global condition keys: aws:SourceVpc

Error – Invalid vpce format

Issue code: INVALID_VPCE_FORMAT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid vpce format: The VPCE identifier in the condition key value is not valid.  Use
 the prefix 'vpce-' followed by 8 or 17 alphanumeric characters.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The VPCE identifier in the condition key value is not valid.  Use
 the prefix 'vpce-' followed by 8 or 17 alphanumeric characters."
```

Resolving the error

The `aws:SourceVpce` condition key must use the prefix `vpce-` followed by either 8 or 17 alphanumeric characters, for example, `vpce-11223344556677889` or `vpce-12345678`.

Related terms

- AWS global condition keys: aws:SourceVpce

Error – Federated principal not supported

Issue code: FEDERATED_PRINCIPAL_NOT_SUPPORTED

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Federated principal not supported: The policy type does not support a federated
 identity provider in the principal element. Use a supported principal.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The policy type does not support a federated identity provider in
 the principal element. Use a supported principal."
```

Resolving the error

The `Principal` element uses federated principals for trust policies attached to IAM roles to provide access through identity federation. Identity policies and other resource-based policies don't support a federated identity provider in the `Principal` element. For example, you can't use a SAML principal in an Amazon S3 bucket policy. Change the `Principal` element to a supported principal type.

Related terms

- Creating a role for identity federation
- JSON policy elements: Principal

Error – Unsupported action for condition key

Issue code: UNSUPPORTED_ACTION_FOR_CONDITION_KEY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported action for condition key: The following actions: {{actions}} are not
 supported by the condition key {{key}}. The condition will not be evaluated for these
 actions. We recommend that you move these actions to a different statement without
 this condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The following actions: {{actions}} are not supported by the
 condition key {{key}}. The condition will not be evaluated for these actions. We
 recommend that you move these actions to a different statement without this condition
 key."
```

Resolving the error

Make sure that the condition key in the `Condition` element of the policy statement applies to every action in the `Action` element. To ensure that the actions you specify are effectively allowed

or denied by your policy, you should move the unsupported actions to a different statement without the condition key.

> **Note**
>
> If the `Action` element has actions with wildcards, IAM Access Analyzer doesn't evaluate those actions for this error.

Related terms

- JSON policy elements: Action

Error – Unsupported action in policy

Issue code: UNSUPPORTED_ACTION_IN_POLICY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported action in policy: The action {{action}} is not supported for the resource-
based policy attached to the resource type {{resourceType}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The action {{action}} is not supported for the resource-based policy
 attached to the resource type {{resourceType}}."
```

Resolving the error

Some actions aren't supported in the `Action` element in the resource-based policy attached to a different resource type. For example, AWS Key Management Service actions aren't supported in Amazon S3 bucket policies. Specify an action that is supported by resource type attached to your resource-based policy.

Related terms

- JSON policy elements: Action

Error – Unsupported resource ARN in policy

Issue code: UNSUPPORTED_RESOURCE_ARN_IN_POLICY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported resource ARN in policy: The resource ARN is not supported for the resource-based policy attached to the resource type {{resourceType}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource ARN is not supported for the resource-based policy attached to the resource type {{resourceType}}."
```

Resolving the error

Some resource ARNs aren't supported in the `Resource` element of the resource-based policy when the policy is attached to a different resource type. For example, AWS KMS ARNs aren't supported in the `Resource` element for Amazon S3 bucket policies. Specify a resource ARN that is supported by a resource type attached to your resource-based policy.

Related terms

- JSON policy elements: Action

Error – Unsupported condition key for service principal

Issue code: UNSUPPORTED_CONDITION_KEY_FOR_SERVICE_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unsupported condition key for service principal: The following condition keys are not
 supported when used with the service principal: {{conditionKeys}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The following condition keys are not supported when used with the
 service principal: {{conditionKeys}}."
```

Resolving the error

You can specify AWS services in the `Principal` element of a resource-based policy using a *service principal*, which is an identifier for the service. You can't use some condition keys with certain service principals. For example, you can't use the `aws:PrincipalOrgID` condition key with the service principal `cloudfront.amazonaws.com`. You should remove condition keys that do not apply to the service principal in the `Principal` element.

Related terms

- Service principals
- JSON policy elements: Principal

Error – Role trust policy syntax error notprincipal

Issue code: ROLE_TRUST_POLICY_SYNTAX_ERROR_NOTPRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Role trust policy syntax error notprincipal: Role trust policies do not support
 NotPrincipal. Update the policy to use a Principal element instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Role trust policies do not support NotPrincipal. Update the policy
 to use a Principal element instead."
```

Resolving the error

A role trust policy is a resource-based policy that is attached to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. Role trust policies do not support `NotPrincipal`. Update the policy to use a `Principal` element instead.

Related terms

- JSON policy elements: Principal
- JSON policy elements: NotPrincipal

Error – Role trust policy unsupported wildcard in principal

Issue code: ROLE_TRUST_POLICY_UNSUPPORTED_WILDCARD_IN_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Role trust policy unsupported wildcard in principal: "Principal:" "*" is not supported
 in the principal element of a role trust policy. Replace the wildcard with a valid
 principal value.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": ""Principal:" "*" is not supported in the principal element of a role
 trust policy. Replace the wildcard with a valid principal value."
```

Resolving the error

A role trust policy is a resource-based policy that is attached to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role.

`"Principal:" "*"` is not supported in the `Principal` element of a role trust policy. Replace the wildcard with a valid principal value.

Related terms

- JSON policy elements: Principal

Error – Role trust policy syntax error resource

Issue code: ROLE_TRUST_POLICY_SYNTAX_ERROR_RESOURCE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Role trust policy syntax error resource: Role trust policies apply to the role
 that they are attached to. You cannot specify a resource. Remove the Resource or
 NotResource element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Role trust policies apply to the role that they are attached to. You
 cannot specify a resource. Remove the Resource or NotResource element."
```

Resolving the error

A role trust policy is a resource-based policy that is attached to an IAM role. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role. Role trust policies apply to the role that they are attached to. You cannot specify a `Resource` or `NotResource` element in a role trust policy. Remove the `Resource` or `NotResource` element.

- JSON policy elements: Resource
- JSON policy elements: NotResource

Error – Type mismatch IP range

Issue code: TYPE_MISMATCH_IP_RANGE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch IP range: The condition operator {{operator}} is used with an invalid IP
 range value. Specify the IP range in standard CIDR format.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition operator {{operator}} is used with an invalid IP range
 value. Specify the IP range in standard CIDR format."
```

Resolving the error

Update the text to use the IP address condition operator data type, in a CIDR format.

Related terms

- IP address condition operators
- IAM JSON policy elements: Condition operators

Error – Missing action for condition key

Issue code: MISSING_ACTION_FOR_CONDITION_KEY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing action for condition key: The {{actionName}} action must be in the action block
 to allow setting values for the condition key {{keyName}}. Add {{actionName}} to the
 action block.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{actionName}} action must be in the action block to allow
 setting values for the condition key {{keyName}}. Add {{actionName}} to the action
 block."
```

Resolving the error

The condition key in the `Condition` element of the policy statement is not evaluated unless the specified action is in the `Action` element. To ensure that the condition keys you specify are effectively allowed or denied by your policy, add the action to the `Action` element.

Related terms

- JSON policy elements: Action

Error – Invalid federated principal syntax in role trust policy

Issue code: INVALID_FEDERATED_PRINCIPAL_SYNTAX_IN_ROLE_TRUST_POLICY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid federated principal syntax in role trust policy: The principal value specifies
 a federated principal that does not match the expected format. Update the federated
 principal to a domain name or a SAML metadata ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The principal value specifies a federated principal that does not
 match the expected format. Update the federated principal to a domain name or a SAML
 metadata ARN."
```

Resolving the error

The principal value specifies a federated principal that does not match the expected format. Update the format of the federated principal to a valid domain name or a SAML metadata ARN.

Related terms

- Federated users and roles

Error – Mismatched action for principal

Issue code: MISMATCHED_ACTION_FOR_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Mismatched action for principal: The {{actionName}} action is invalid with the
 following principal(s): {{principalNames}}. Use a SAML provider principal with
 the sts:AssumeRoleWithSAML action or use an OIDC provider principal with the
 sts:AssumeRoleWithWebIdentity action. Ensure the provider is Federated if you use
 either of the two options.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{actionName}} action is invalid with the following
 principal(s): {{principalNames}}. Use a SAML provider principal with the
 sts:AssumeRoleWithSAML action or use an OIDC provider principal with the
 sts:AssumeRoleWithWebIdentity action. Ensure the provider is Federated if you use
 either of the two options."
```

Resolving the error

The action specified in the `Action` element of the policy statement is invalid with the principal specified in the `Principal` element. For example, you can't use a SAML provider principal with the `sts:AssumeRoleWithWebIdentity` action. You should use a SAML provider principal with the `sts:AssumeRoleWithSAML` action or use an OIDC provider principal with the `sts:AssumeRoleWithWebIdentity` action.

Related terms

- AssumeRoleWithSAML
- AssumeRoleWithWebIdentity

Error – Missing action for roles anywhere trust policy

Issue code: MISSING_ACTION_FOR_ROLES_ANYWHERE_TRUST_POLICY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing action for roles anywhere trust policy: The rolesanywhere.amazonaws.com service
 principal requires the sts:AssumeRole, sts:SetSourceIdentity, and sts:TagSession
 permissions to assume a role. Add the missing permissions to the policy.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The rolesanywhere.amazonaws.com service principal requires the
 sts:AssumeRole, sts:SetSourceIdentity, and sts:TagSession permissions to assume a
 role. Add the missing permissions to the policy."
```

Resolving the error

For IAM Roles Anywhere to be able to assume a role and deliver temporary AWS credentials, the role must trust the IAM Roles Anywhere service principal. The IAM Roles Anywhere service principal requires the `sts:AssumeRole`, `sts:SetSourceIdentity`, and `sts:TagSession` permissions to assume a role. If any of the permissions are missing, you must add them to your policy.

Related terms

- Trust model in AWS Identity and Access Management Roles Anywhere

Error – Policy size exceeds RCP quota

Issue code: POLICY_SIZE_EXCEEDS_RCP_QUOTA

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Policy size exceeds RCP quota: The {{policySize}} characters in the resource control
 policy (RCP) exceed the {{policySizeQuota}} character maximum for RCPs. We recommend
 that you use multiple granular policies.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{policySize}} characters in the resource control policy (RCP)
 exceed the {{policySizeQuota}} character maximum for RCPs. We recommend that you use
 multiple granular policies."
```

Resolving the error

AWS Organizations resource control policies (RCPs) support specifying values in the `Action` element. However, these values can include wildcards (*) only at the end of the string. This means that you can specify `s3:Get*` but not `s3:*Object`.

To specify multiple actions, AWS recommends that you list them individually.

Related terms

- Quotas for AWS Organizations
- AWS Organizations resource control policies

Error – RCP syntax error principal

Issue code: RCP_SYNTAX_ERROR_PRINCIPAL

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
RCP syntax error principal: The Principal element contents are not valid. RCPs only
 support specifying all principals ("*") in the Principal element. The NotPrincipal
 element is not supported for RCPs.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Principal element contents are not valid. RCPs only support
 specifying all principals ("*") in the Principal element. The NotPrincipal element is
 not supported for RCPs."
```

Resolving the error

AWS Organizations resource control policies (RCPs) only support specifying all principals ("*") in the `Principal` element. The `NotPrincipal` element is not supported for RCPs.

Related terms

- RCP syntax
- Properties of the principal

Error – RCP syntax error allow

Issue code: RCP_SYNTAX_ERROR_ALLOW

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
RCP syntax error allow: RCPs only support specifying all principals ("*") in the
 Principal element, all resources ("*") in the Resource element, and no Condition
 element with an effect of Allow.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "RCPs only support specifying all principals ("*") in the Principal
 element, all resources ("*") in the Resource element, and no Condition element with an
 effect of Allow."
```

Resolving the error

AWS Organizations resource control policies (RCPs) only support specifying all principals ("*") in the `Principal` element and all resources ("*") in the `Resource` element. The `Condition` element with an effect of `Allow` is not supported for RCPs.

Related terms

- RCP syntax
- Properties of the principal
- Properties of the resource

Error – RCP syntax error NotAction

Issue code: RCP_SYNTAX_ERROR_NOTACTION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
RCP syntax error NotAction: RCPs do not support the NotAction element. Update to use
 the Action element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "RCPs do not support the NotAction element. Update to use the Action
 element."
```

Resolving the error

AWS Organizations resource control policies (RCPs) do not support the `NotAction` element. Use the `Action` element.

Related terms

- RCP syntax
- IAM JSON policy elements: Action
- IAM JSON policy elements: NotAction

Error – RCP syntax error action

Issue code: RCP_SYNTAX_ERROR_ACTION

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
RCP syntax error action: RCPs only support specifying select service prefixes in the
 Action element. Learn more here.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "RCPs only support specifying select service prefixes in the Action
 element. Learn more here."
```

Resolving the error

AWS Organizations resource control policies (RCPs) only support specifying select service prefixes in the `Action` element.

Related terms

- RCP syntax
- List of AWS services that support RCPs

Error – Missing ARN account

Issue code: MISSING_ARN_ACCOUNT

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing ARN account: The resource {{resourceName}} in the arn is missing an account id.
 Please provide a 12 digit account id.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The resource {{resourceName}} in the arn is missing an account id.
 Please provide a 12 digit account id."
```

Resolving the error

Include an account ID in the resource ARN. Account IDs are 12-digit integers. To learn how to view your account ID, see Finding your AWS account ID.

Related terms

- Policy resources
- Account Identifiers
- Resource ARNs
- AWS service resources with ARN formats

Error – Invalid kms key value

Issue code: INVALID_KMS_KEY_VALUE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid kms key value: The {{key}} condition key value must be a valid KMS key ARN.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{key}} condition key value must be a valid KMS key ARN."
```

Resolving the error

An AWS KMS key ARN (Amazon Resource Name) is a unique, fully qualified identifier for a KMS key. A key ARN includes the AWS account, Region, and the key ID. A key ARN follows this format:

`arn:aws:kms:`*`region`*`:`*`account-id`*`:key/`*`key-id`*

Related terms

- Find the key ID and key ARN
- Key ARN
- AWS global condition context keys

Error – Variable usage too permissive

Issue code: VARIABLE_USAGE_TOO_PERMISSIVE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Variable usage too permissive: Overly permissive use of policy variable for the {{key}}
 condition key. Use the policy variable preceded by 6 consecutive characters.
```

```
The policy variable is not allowed in the condition key {{key}}. We consider the key to
 be sensitive and policy variables can be evaluated as effective wildcards. Therefore
 policy variables are not allowed to be used with sensitive keys. Refer to https://
docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_condition-keys.html for the
 list of sensitive keys.
```

```
Overly permissive use of policy variable with aws:userID. Use the policy variable on
 the right side of colon or as the only character on the left side of a colon.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Overly permissive use of policy variable for the {{key}} condition
 key. Use the policy variable preceded by 6 consecutive characters."
```

```
"findingDetails": "The policy variable is not allowed in the condition key {{key}}.
 We consider the key to be sensitive and policy variables can be evaluated as
 effective wildcards. Therefore policy variables are not allowed to be used
 with sensitive keys. Refer to https://docs.aws.amazon.com/IAM/latest/UserGuide/
reference_policies_condition-keys.html for the list of sensitive keys."
```

```
"findingDetails": "Overly permissive use of policy variable with aws:userID. Use the
 policy variable on the right side of colon or as the only character on the left side
 of a colon."
```

Resolving the error

There are three possible finding messages for this error.

- For the first error message, modify your policy variable usage to be more specific. Add at least 6 consecutive characters before the policy variable to reduce the scope of permissions. For example, instead of using `${aws:username}`, use `prefix-${aws:username}` or `myapp-${aws:username}`. This ensures that the policy variable doesn't grant overly broad access.
- For the second error message, remove the policy variable from the specified condition key. Policy variables can act as effective wildcards and are not permitted with sensitive condition keys for security reasons. Instead, use specific static values or consider restructuring your policy to use non-sensitive condition keys that support policy variables.
- For the third error message, modify your `aws:userID` policy variable usage to be more restrictive. Place the policy variable on the right side of a colon (after the account ID) or use it as the only character on the left side of a colon. For example, use `AIDACKCEVSQ6C2EXAMPLE:${aws:userid}` or `${aws:userid}:*` instead of `${aws:userid}`.

Related terms

- IAM policy elements: Variables and tags
- IAM policy elements: Condition
- Conditions with multiple context keys or values
- AWS global condition context keys

Error – Wildcard usage too permissive

Issue code: WILDCARD_USAGE_TOO_PERMISSIVE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Wildcard usage too permissive: Overly permissive use of wildcard for the {{key}}
 condition key. Use the wildcard preceded by 6 consecutive characters.
```

```
The wildcard is not allowed in the condition key {{key}}. We consider the key to
 be sensitive and wildcards are not allowed to be used with sensitive keys. Refer
 to https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_condition-
keys.html for the list of sensitive keys.
```

```
Overly permissive use of wildcard with aws:userID. Use the wildcard on the right side
 of colon or as the only character on the left side of a colon.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Overly permissive use of wildcard for the {{key}} condition key. Use
 the wildcard preceded by 6 consecutive characters."
```

```
"findingDetails": "The wildcard is not allowed in the condition key {{key}}.
 We consider the key to be sensitive and wildcards are not allowed to be used
 with sensitive keys. Refer to https://docs.aws.amazon.com/IAM/latest/UserGuide/
reference_policies_condition-keys.html for the list of sensitive keys."
```

```
"findingDetails": "Overly permissive use of wildcard with aws:userID. Use the wildcard
 on the right side of colon or as the only character on the left side of a colon."
```

Resolving the error

There are three possible finding messages for this error.

- For the first error message, make your wildcard usage more specific by adding at least 6 consecutive characters before the wildcard. For example, instead of using `*`, use `prefix-*` or `prefix-*-suffix`. This reduces the scope of the condition and follows the principle of least privilege.
- For the second error message, remove the wildcard from the specified condition key. Wildcards are not permitted with sensitive condition keys for security reasons. Replace the wildcard with specific values that match your intended access pattern, or consider using a different, non-sensitive condition key that supports wildcards.

- For the third error message, modify your `aws:userID` wildcard usage to be more restrictive. Place the wildcard on the right side of a colon (after the account ID) or use it as the only character on the left side of a colon. For example, use `AIDACKCEVSQ6C2EXAMPLE:*` or `*:*` instead of `*`.

Related terms

- IAM policy elements: Condition
- Conditions with multiple context keys or values
- AWS global condition context keys
- IAM identifiers

Error – DynamoDB invalid return consumed capacity

Issue code: DYNAMODB_INVALID_RETURN_CONSUMED_CAPACITY

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB invalid return consumed capacity: The value for
 dynamodb:ReturnConsumedCapacity is not valid. Update {{value}} to either TOTAL or
 NONE.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The value for dynamodb:ReturnConsumedCapacity is not valid. Update
 {{value}} to either TOTAL or NONE."
```

Resolving the error

Update the condition value to one of the two values DynamoDB provides: `TOTAL` or `NONE`. Values are case-sensitive under `StringEquals` and `StringLike`, so a lowercase variant such as `total` is not valid. The `StringEqualsIgnoreCase` operator accepts any casing. A condition that specifies an invalid value never matches any request, so the statement has no effect.

Error – DynamoDB invalid return values

Issue code: DYNAMODB_INVALID_RETURN_VALUES

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB invalid return values: The value for dynamodb:ReturnValues is not valid.
 Update {{value}} to one of NONE, ALL_OLD, UPDATED_OLD, ALL_NEW, or UPDATED_NEW.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The value for dynamodb:ReturnValues is not valid. Update {{value}}
 to one of NONE, ALL_OLD, UPDATED_OLD, ALL_NEW, or UPDATED_NEW."
```

Resolving the error

Update the condition value to one of the five values DynamoDB provides: `NONE`, `ALL_OLD`, `UPDATED_OLD`, `ALL_NEW`, or `UPDATED_NEW`. Values are case-sensitive under `StringEquals` and `StringLike`, so a lowercase variant such as `updated_new` is not valid. The `StringEqualsIgnoreCase` operator accepts any casing. A condition that specifies an invalid value never matches any request, so the statement has no effect.

Error – DynamoDB invalid select value

Issue code: DYNAMODB_INVALID_SELECT_VALUE

Finding type: ERROR

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB invalid select value: The value for dynamodb:Select is not valid. Update
 {{value}} to one of ALL_ATTRIBUTES, ALL_PROJECTED_ATTRIBUTES, SPECIFIC_ATTRIBUTES, or
 COUNT.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The value for dynamodb:Select is not valid. Update {{value}} to one
 of ALL_ATTRIBUTES, ALL_PROJECTED_ATTRIBUTES, SPECIFIC_ATTRIBUTES, or COUNT."
```

Resolving the error

Update the condition value to one of the four values DynamoDB provides: `ALL_ATTRIBUTES`, `ALL_PROJECTED_ATTRIBUTES`, `SPECIFIC_ATTRIBUTES`, or `COUNT`. Values are case-sensitive under `StringEquals` and `StringLike`, so a lowercase variant such as `specific_attributes` is not valid. The `StringEqualsIgnoreCase` operator accepts any casing. A condition that specifies an invalid value never matches any request, so the statement has no effect.

General Warning – Create SLR with NotResource

Issue code: CREATE_SLR_WITH_NOT_RESOURCE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Create SLR with NotResource: Using the iam:CreateServiceLinkedRole action with
 NotResource can allow creation of unintended service-linked roles for multiple
 resources. We recommend that you specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the iam:CreateServiceLinkedRole action with NotResource can
 allow creation of unintended service-linked roles for multiple resources. We recommend
 that you specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using `iam:CreateServiceLinkedRole` in a policy with the `NotResource` element can allow creating unintended service-linked roles for

multiple resources. AWS recommends that you specify allowed ARNs in the `Resource` element instead.

- CreateServiceLinkedRole operation
- IAM JSON policy elements: NotResource
- IAM JSON policy elements: Resource

General Warning – Create SLR with star in action and NotResource

Issue code: CREATE_SLR_WITH_STAR_IN_ACTION_AND_NOT_RESOURCE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Create SLR with star in action and NotResource: Using an action with a wildcard(*) and
 NotResource can allow creation of unintended service-linked roles because it can allow
 iam:CreateServiceLinkedRole permissions on multiple resources. We recommend that you
 specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using an action with a wildcard(*) and NotResource can
 allow creation of unintended service-linked roles because it can allow
 iam:CreateServiceLinkedRole permissions on multiple resources. We recommend that you
 specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Policies with a wildcard (*) in the `Action` and that include the `NotResource` element can allow creation of unintended service-linked roles for multiple resources. AWS recommends that you specify allowed ARNs in the `Resource` element instead.

- CreateServiceLinkedRole operation

- IAM JSON policy elements: NotResource
- IAM JSON policy elements: Resource

General Warning – Create SLR with NotAction and NotResource

Issue code: CREATE_SLR_WITH_NOT_ACTION_AND_NOT_RESOURCE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Create SLR with NotAction and NotResource: Using NotAction with NotResource
 can allow creation of unintended service-linked roles because it allows
 iam:CreateServiceLinkedRole permissions on multiple resources. We recommend that you
 specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using NotAction with NotResource can allow creation of unintended
 service-linked roles because it allows iam:CreateServiceLinkedRole permissions on
 multiple resources. We recommend that you specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using the `NotAction` element with the `NotResource` element can allow creating unintended service-linked roles for multiple resources. AWS recommends that you rewrite the policy to allow `iam:CreateServiceLinkedRole` on a limited list of ARNs in the `Resource` element instead. You can also add `iam:CreateServiceLinkedRole` to the `NotAction` element.

- CreateServiceLinkedRole operation
- IAM JSON policy elements: NotAction
- IAM JSON policy elements: Action
- IAM JSON policy elements: NotResource

- IAM JSON policy elements: Resource

General Warning – Create SLR with star in resource

Issue code: CREATE_SLR_WITH_STAR_IN_RESOURCE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Create SLR with star in resource: Using the iam:CreateServiceLinkedRole action with
 wildcards (*) in the resource can allow creation of unintended service-linked roles.
 We recommend that you specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the iam:CreateServiceLinkedRole action with wildcards (*) in
 the resource can allow creation of unintended service-linked roles. We recommend that
 you specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using `iam:CreateServiceLinkedRole` in a policy with a wildcard (*) in the `Resource` element can allow creating unintended service-linked roles for multiple resources. AWS recommends that you specify allowed ARNs in the `Resource` element instead.

- CreateServiceLinkedRole operation
- IAM JSON policy elements: Resource

AWS managed policies with this general warning

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

Some of those use cases are for power users within your account. The following AWS managed policies provide power user access and grant permissions to create service-linked roles for any AWS service. AWS recommends that you attach the following AWS managed policies to only IAM identities that you consider power users.

- PowerUserAccess
- AlexaForBusinessFullAccess
- AWSOrganizationsServiceTrustPolicy – This AWS managed policy provides permissions for use by the AWS Organizations service-linked role. This role allows Organizations to create additional service-linked roles for other services in your AWS organization.

General Warning – Create SLR with star in action and resource

Issue code: CREATE_SLR_WITH_STAR_IN_ACTION_AND_RESOURCE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Create SLR with star in action and resource: Using wildcards (*) in the action and
 the resource can allow creation of unintended service-linked roles because it allows
 iam:CreateServiceLinkedRole permissions on all resources. We recommend that you
 specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using wildcards (*) in the action and the resource
 can allow creation of unintended service-linked roles because it allows
 iam:CreateServiceLinkedRole permissions on all resources. We recommend that you
 specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Policies with a wildcard (*) in the `Action` and `Resource` elements can allow creating unintended service-linked roles for multiple resources. This

allows creating a service-linked role when you specify `"Action": "*"`, `"Action": "iam:*"`, or `"Action": "iam:Create*"`. AWS recommends that you specify allowed ARNs in the `Resource` element instead.

- CreateServiceLinkedRole operation
- IAM JSON policy elements: Action
- IAM JSON policy elements: Resource

AWS managed policies with this general warning

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

Some of those use cases are for administrators within your account. The following AWS managed policies provide administrator access and grant permissions to create service-linked roles for any AWS service. AWS recommends that you attach the following AWS managed policies to only the IAM identities that you consider administrators.

- AdministratorAccess
- IAMFullAccess

General Warning – Create SLR with star in resource and NotAction

Issue code: CREATE_SLR_WITH_STAR_IN_RESOURCE_AND_NOT_ACTION

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Create SLR with star in resource and NotAction: Using a resource with wildcards (*)
 and NotAction can allow creation of unintended service-linked roles because it allows
 iam:CreateServiceLinkedRole permissions on all resources. We recommend that you
 specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using a resource with wildcards (*) and NotAction can allow creation
 of unintended service-linked roles because it allows iam:CreateServiceLinkedRole
 permissions on all resources. We recommend that you specify resource ARNs instead."
```

Resolving the general warning

The action `iam:CreateServiceLinkedRole` grants permission to create an IAM role that allows an AWS service to perform actions on your behalf. Using the `NotAction` element in a policy with a wildcard (*) in the `Resource` element can allow creating unintended service-linked roles for multiple resources. AWS recommends that you specify allowed ARNs in the `Resource` element instead. You can also add `iam:CreateServiceLinkedRole` to the `NotAction` element.

- CreateServiceLinkedRole operation
- IAM JSON policy elements: NotAction
- IAM JSON policy elements: Action
- IAM JSON policy elements: Resource

General Warning – Deprecated global condition key

Issue code: DEPRECATED_GLOBAL_CONDITION_KEY

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Deprecated global condition key: We recommend that you update aws:ARN to use the newer
 condition key aws:PrincipalArn.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "We recommend that you update aws:ARN to use the newer condition key
 aws:PrincipalArn."
```

Resolving the general warning

The policy includes a deprecated global condition key. Update the condition key in the condition key-value pair to use a supported global condition key.

- Global condition keys

General Warning – Invalid date value

Issue code: INVALID_DATE_VALUE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid date value: The date {{date}} might not resolve as expected. We recommend that
 you use the YYYY-MM-DD format.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The date {{date}} might not resolve as expected. We recommend that
 you use the YYYY-MM-DD format."
```

Resolving the general warning

Unix Epoch time describes a point in time that has elapsed since January 1, 1970, minus leap seconds. Epoch time might not resolve to the precise time that you expect. AWS recommends that you use the W3C standard for date and time formats. For example, you could specify a complete date, such as YYYY-MM-DD (1997-07-16), or you could also append the time to the second, such as YYYY-MM-DDThh:mm:ssTZD (1997-07-16T19:20:30+01:00).

- W3C Date and Time Formats
- IAM JSON policy elements: Version
- aws:CurrentTime global condition key

General Warning – Invalid role reference

Issue code: INVALID_ROLE_REFERENCE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid role reference: The Principal element includes the IAM role ID {{roleid}}. We
 recommend that you use a role ARN instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Principal element includes the IAM role ID {{roleid}}. We
 recommend that you use a role ARN instead."
```

Resolving the general warning

AWS recommends that you specify the Amazon Resource Name (ARN) for an IAM role instead of its principal ID. When IAM saves the policy, it will transform the ARN into the principal ID for the existing role. AWS includes a safety precaution. If someone deletes and recreates the role, it will have a new ID, and the policy won't match the new role's ID.

- Specifying a principal: IAM roles
- IAM ARNs
- IAM unique IDs

General Warning – Invalid user reference

Issue code: INVALID_USER_REFERENCE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Invalid user reference: The Principal element includes the IAM user ID {{userid}}. We
 recommend that you use a user ARN instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The Principal element includes the IAM user ID {{userid}}. We
 recommend that you use a user ARN instead."
```

Resolving the general warning

AWS recommends that you specify the Amazon Resource Name (ARN) for an IAM user instead of its principal ID. When IAM saves the policy, it will transform the ARN into the principal ID for the existing user. AWS includes a safety precaution. If someone deletes and recreates the user, it will have a new ID, and the policy won't match the new user's ID.

- Specifying a principal: IAM users
- IAM ARNs
- IAM unique IDs

General Warning – Missing version

Issue code: MISSING_VERSION

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing version: We recommend that you specify the Version element to help you with
 debugging permission issues.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "We recommend that you specify the Version element to help you with
 debugging permission issues."
```

Resolving the general warning

AWS recommends that you include the optional `Version` parameter in your policy. If you do not include a Version element, the value defaults to `2012-10-17`, but newer features, such as policy

variables, will not work with your policy. For example, variables such as `${aws:username}` aren't recognized as variables and are instead treated as literal strings in the policy.

- IAM JSON policy elements: Version

General Warning – Unique Sids recommended

Issue code: UNIQUE_SIDS_RECOMMENDED

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Unique Sids recommended: We recommend that you use statement IDs that are unique to
 your policy. Update the Sid value.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "We recommend that you use statement IDs that are unique to your
 policy. Update the Sid value."
```

Resolving the general warning

AWS recommends that you use unique statement IDs. The `Sid` (statement ID) element allows you to enter an optional identifier that you provide for the policy statement. You can assign a statement ID value to each statement in a statement array using the `SID` element.

Related terms

- IAM JSON policy elements: Sid

General Warning – Wildcard without like operator

Issue code: WILDCARD_WITHOUT_LIKE_OPERATOR

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Wildcard without like operator: Your condition value includes a * or ? character. If
 you meant to use a wildcard (*, ?), update the condition operator to include Like.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Your condition value includes a * or ? character. If you meant to
 use a wildcard (*, ?), update the condition operator to include Like."
```

Resolving the general warning

The `Condition` element structure requires that you use a condition operator and a key-value pair. When you specify a condition value that uses a wildcard (*, ?), you must use the `Like` version of the condition operator. For example, instead of the `StringEquals` string condition operator, use `StringLike`.

```
"Condition": {"StringLike": {"aws:PrincipalTag/job-category": "admin-*"}}
```

- IAM JSON policy elements: Condition operators
- IAM JSON policy elements: Condition

AWS managed policies with this general warning

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

The following AWS managed policies include wildcards in their condition value without a condition operator that includes `Like` for pattern-matching. When using the AWS managed policy as a reference to create your customer managed policy, AWS recommends that you use a condition operator that supports pattern-matching with wildcards (*, ?), such as `StringLike`.

- AWSGlueConsoleSageMakerNotebookFullAccess

General Warning – Policy size exceeds identity policy quota

Issue code: POLICY_SIZE_EXCEEDS_IDENTITY_POLICY_QUOTA

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Policy size exceeds identity policy quota: The {{policySize}} characters in the
 identity policy, excluding whitespace, exceed the {{policySizeQuota}} character
 maximum for inline and managed policies. We recommend that you use multiple granular
 policies.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{policySize}} characters in the identity policy, excluding
 whitespace, exceed the {{policySizeQuota}} character maximum for inline and managed
 policies. We recommend that you use multiple granular policies."
```

Resolving the general warning

You can attach up to 20 managed policies to an IAM identity (user, group of users, or role). However, the size of each managed policy cannot exceed the default quota of 6,144 characters. IAM does not count white space when calculating the size of a policy against this quota. Quotas, also referred to as limits in AWS, are the maximum values for the resources, actions, and items in your AWS account.

Additionally, you can add as many inline policies as you want to an IAM identity. However, the sum size of all inline policies per identity cannot exceed the specified quota.

If your policy is larger than the quota, you can organize your policy into multiple statements and group the statements into multiple policies.

Related terms

- IAM and AWS STS character quotas
- Multiple statements and multiple policies
- IAM customer managed policies
- Overview of JSON policies
- IAM JSON policy grammar

AWS managed policies with this general warning

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

The following AWS managed policies grant permissions to actions across many AWS services and exceed the maximum policy size. When using the AWS managed policy as a reference to create your managed policy, you must split the policy into multiple policies.

- ReadOnlyAccess
- AWSSupportServiceRolePolicy

General Warning – Policy size exceeds resource policy quota

Issue code: POLICY_SIZE_EXCEEDS_RESOURCE_POLICY_QUOTA

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Policy size exceeds resource policy quota: The {{policySize}} characters in the
 resource policy exceed the {{policySizeQuota}} character maximum for resource
 policies. We recommend that you use multiple granular policies.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{policySize}} characters in the resource policy exceed the
 {{policySizeQuota}} character maximum for resource policies. We recommend that you use
 multiple granular policies."
```

Resolving the general warning

Resource-based policies are JSON policy documents that you attach to a resource, such as an Amazon S3 bucket. These policies grant the specified principal permission to perform specific actions on that resource and define under what conditions this applies. The size of resource-based policies cannot exceed the quota set for that resource. Quotas, also referred to as limits in AWS, are the maximum values for the resources, actions, and items in your AWS account.

If your policy is larger than the quota, you can organize your policy into multiple statements and group the statements into multiple policies.

Related terms

- Resource-based policies
- Amazon S3 bucket policies
- Multiple statements and multiple policies
- Overview of JSON policies
- IAM JSON policy grammar

General Warning – Type mismatch

Issue code: TYPE_MISMATCH

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch: Use the operator type {{allowed}} instead of operator {{operator}} for
 the condition key {{key}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Use the operator type {{allowed}} instead of operator {{operator}}
 for the condition key {{key}}."
```

Resolving the general warning

Update the text to use the supported condition operator data type.

For example, the `aws:MultiFactorAuthPresent` global condition key requires a condition operator with the `Boolean` data type. If you provide a date or an integer, the data type won't match.

Related terms

- Global condition keys
- IAM JSON policy elements: Condition operators

General Warning – Type mismatch Boolean

Issue code: TYPE_MISMATCH_BOOLEAN

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch Boolean: Add a valid Boolean value (true or false) for the condition
 operator {{operator}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a valid Boolean value (true or false) for the condition operator
 {{operator}}."
```

Resolving the general warning

Update the text to use a Boolean condition operator data type, such as `true` or `false`.

For example, the `aws:MultiFactorAuthPresent` global condition key requires a condition operator with the `Boolean` data type. If you provide a date or an integer, the data type won't match.

Related terms

- Boolean condition operators
- IAM JSON policy elements: Condition operators

General Warning – Type mismatch date

Issue code: TYPE_MISMATCH_DATE

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch date: The date condition operator is used with an invalid value. Specify
 a valid date using YYYY-MM-DD or other ISO 8601 date/time format.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The date condition operator is used with an invalid value. Specify a
 valid date using YYYY-MM-DD or other ISO 8601 date/time format."
```

Resolving the general warning

Update the text to use the date condition operator data type, in a `YYYY-MM-DD` or other ISO 8601 date time format.

Related terms

- Date condition operators
- IAM JSON policy elements: Condition operators

General Warning – Type mismatch number

Issue code: TYPE_MISMATCH_NUMBER

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch number: Add a valid numeric value for the condition operator
 {{operator}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a valid numeric value for the condition operator {{operator}}."
```

Resolving the general warning

Update the text to use the numeric condition operator data type.

Related terms

- Numeric condition operators
- IAM JSON policy elements: Condition operators

General Warning – Type mismatch string

Issue code: TYPE_MISMATCH_STRING

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Type mismatch string: Add a valid base64-encoded string value for the condition
 operator {{operator}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a valid base64-encoded string value for the condition operator
 {{operator}}."
```

Resolving the general warning

Update the text to use the string condition operator data type.

Related terms

- String condition operators
- IAM JSON policy elements: Condition operators

General Warning – Specific github repo and branch recommended

Issue code: SPECIFIC_GITHUB_REPO_AND_BRANCH_RECOMMENDED

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Specific github repo and branch recommended: Using a wildcard (*) in
 token.actions.githubusercontent.com:sub can allow requests from more sources than
 you intended. Specify the value of token.actions.githubusercontent.com:sub with the
 repository and branch name. If the subject claim does not include the branch, specify
 it in a token.actions.githubusercontent.com:ref condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using a wildcard (*) in token.actions.githubusercontent.com:sub
 can allow requests from more sources than you intended. Specify the value
 of token.actions.githubusercontent.com:sub with the repository and branch
 name. If the subject claim does not include the branch, specify it in a
 token.actions.githubusercontent.com:ref condition key."
```

Resolving the general warning

If you use GitHub as an OIDC IdP, best practice is to limit the entities that can assume the role associated with the IAM IdP. When you include a `Condition` statement in a role trust policy, you can limit the role to a specific GitHub organization, repository, or branch. You can use the condition key `token.actions.githubusercontent.com:sub` to limit access. We recommend that you limit the condition to a specific set of repositories or branches. If you use a wildcard (`*`) in `token.actions.githubusercontent.com:sub`, then GitHub Actions from organizations or repositories outside of your control are able to assume roles associated with the GitHub IAM IdP in your AWS account.

Specify the repository in `token.actions.githubusercontent.com:sub`. You can specify the branch in that same value, or in a separate `token.actions.githubusercontent.com:ref` condition key. When the subject claim does not include the branch — for example, when it names a GitHub environment — specify the branch in `token.actions.githubusercontent.com:ref`.

Related terms

- Configuring a role for GitHub OIDC identity provider

General Warning – Policy size exceeds role trust policy quota

Issue code: POLICY_SIZE_EXCEEDS_ROLE_TRUST_POLICY_QUOTA

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Policy size exceeds role trust policy quota: The characters in the role trust policy,
 excluding whitespace, exceed the character maximum. We recommend that you request a
 role trust policy length quota increase using Service Quotas and AWS Support Center.
 If the quotas have already been increased, then you can ignore this warning.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The characters in the role trust policy, excluding whitespace,
 exceed the character maximum. We recommend that you request a role trust policy length
 quota increase using Service Quotas and AWS Support Center. If the quotas have already
 been increased, then you can ignore this warning."
```

Resolving the general warning

IAM and AWS STS have quotas that limit the size of role trust policies. The characters in the role trust policy, excluding whitespace, exceed the character maximum. We recommend that you request a role trust policy length quota increase using Service Quotas and the AWS Support Center Console.

Related terms

- IAM and AWS STS quotas, name requirements, and character limits

General Warning – RCP missing related principal condition key

Issue code: RCP_MISSING_RELATED_PRINCIPAL_CONDITION_KEY

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
RCP missing related principal condition key: RCPs impact IAM roles, users, and AWS
 service principals. To prevent unintended impact to services acting on your behalf
 using a service principal, an additional statement should be added to the Condition
 block "BoolIfExists": { "aws:PrincipalIsAWSService": "false"} whenever a principal key
 {{conditionKeyName}} is used.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "RCPs impact IAM roles, users, and AWS service principals. To prevent
 unintended impact to services acting on your behalf using a service principal,
 an additional statement should be added to the Condition block "BoolIfExists":
 { "aws:PrincipalIsAWSService": "false"} whenever a principal key {{conditionKeyName}}
 is used."
```

Resolving the general warning

AWS Organizations resource control policies (RCPs) can impact IAM roles, users, and AWS service principals. To prevent unintended impact to services acting on your behalf using a service principal, add the following statement to your `Condition` element:

```
"BoolIfExists": { "aws:PrincipalIsAWSService": "false"}
```

Related terms

- RCP syntax
- Properties of the principal

General Warning – RCP missing related service principal condition key

Issue code: RCP_MISSING_RELATED_SERVICE_PRINCIPAL_CONDITION_KEY

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
RCP missing related service principal condition key: RCPs impact IAM roles, users,
 and AWS service principals. To prevent unintended impact to your principals,
 an additional statement should be added to the Condition block "BoolIfExists":
 { "aws:PrincipalIsAWSService": "true"} whenever the key {{conditionKeyName}} is used.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "RCPs impact IAM roles, users, and AWS service principals. To prevent
 unintended impact to your principals, an additional statement should be added to the
 Condition block "BoolIfExists": { "aws:PrincipalIsAWSService": "true"} whenever the
 key {{conditionKeyName}} is used."
```

Resolving the general warning

AWS Organizations resource control policies (RCPs) can impact IAM roles, users, and AWS service principals. To prevent unintended impact to your principals, add the following statement to your `Condition` element:

```
"BoolIfExists": { "aws:PrincipalIsAWSService": "true"}
```

Related terms

- RCP syntax
- Properties of the principal

General Warning – RCP missing service condition key null check

Issue code: RCP_MISSING_SERVICE_CONDITION_KEY_NULL_CHECK

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
RCP missing service condition key null check: The specified service may have a service
 integration that does not require the use of the {{conditionKeyName}} condition
 key. To prevent unintended impact to services acting on your behalf using a service
 principal, an additional statement should be added to the Condition block "Null":
 { "aws:SourceAccount": "false"} or "Null": { "aws:SourceArn": "false"} whenever the
 key {{conditionKeyName}} is used.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The specified service may have a service integration that does
 not require the use of the {{conditionKeyName}} condition key. To prevent unintended
 impact to services acting on your behalf using a service principal, an additional
 statement should be added to the Condition block "Null": { "aws:SourceAccount":
 "false"} or "Null": { "aws:SourceArn": "false"} whenever the key {{conditionKeyName}}
 is used."
```

Resolving the general warning

AWS Organizations resource control policies (RCPs) can impact IAM roles, users, and AWS service principals. To prevent unintended impact to services acting on your behalf using a service principal, add one of the following statements to your `Condition` element whenever the specified key is used:

```
"Null": { "aws:SourceAccount": "false"}
```

or

```
"Null": { "aws:SourceArn": "false"}
```

Related terms

- RCP syntax
- Properties of the principal

General Warning – Use condition key only with supported services

Issue code: USE_CONDITION_KEY_ONLY_WITH_SUPPORTED_SERVICES

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Use condition key only with supported services: The condition key {{key}} works only
 with specific AWS services and must be scoped to supported services in your policies.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{key}} works only with specific AWS services and
 must be scoped to supported services in your policies.
```

Resolving the general warning

Review the AWS documentation to identify which AWS services support this condition key. If any services in your policy don't support the condition key, modify your policy to scope the condition key to only the AWS services that support it.

Related terms

- `aws:VpceAccount`
- `aws:VpceOrgID`
- `aws:VpceOrgPaths`
- Actions, resources, and condition keys for AWS services

General Warning – DynamoDB replace operation with attributes

Issue code: DYNAMODB_REPLACE_OPERATION_WITH_ATTRIBUTES

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB replace operation with attributes: The {{action}} action replaces the entire
 item, so dynamodb:Attributes cannot restrict access at the attribute level. That
```

```
condition only applies to UpdateItem; use dynamodb:UpdateItem for attribute-level
access control.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{action}} action replaces the entire item, so
 dynamodb:Attributes cannot restrict access at the attribute level. That condition only
 applies to UpdateItem; use dynamodb:UpdateItem for attribute-level access control."
```

Resolving the general warning

The dynamodb:Attributes condition does not limit dynamodb:PutItem or dynamodb:BatchWriteItem. These actions replace the whole item, so a caller can drop a restricted attribute just by leaving it out of the item they write. To control which attributes a caller can change, grant dynamodb:UpdateItem instead and keep the dynamodb:Attributes condition on that statement. If you did not intend attribute-level control here, remove the dynamodb:Attributes condition.

General Warning – DynamoDB scan bypasses leading keys

Issue code: DYNAMODB_SCAN_BYPASSES_LEADING_KEYS

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB scan bypasses leading keys: The dynamodb:Scan operation returns all items
 in the table, bypassing the dynamodb:LeadingKeys restriction. Consider removing
 dynamodb:Scan.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The dynamodb:Scan operation returns all items in the table,
 bypassing the dynamodb:LeadingKeys restriction. Consider removing dynamodb:Scan."
```

Resolving the general warning

dynamodb:Scan reads the entire table, so the dynamodb:LeadingKeys condition in this statement does not limit which items a caller can read. Remove dynamodb:Scan, or grant it in a separate statement that does not rely on dynamodb:LeadingKeys.

General Warning – DynamoDB wrong condition for action

Issue code: DYNAMODB_WRONG_CONDITION_FOR_ACTION

Finding type: GENERAL_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB wrong condition for action: The condition key {{conditionKey}} does
 not apply to the {{action}} action. Use dynamodb:Select with read actions and
 dynamodb:ReturnValues with write actions.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{conditionKey}} does not apply to the {{action}}
 action. Use dynamodb:Select with read actions and dynamodb:ReturnValues with write
 actions."
```

Resolving the general warning

This statement uses dynamodb:Select with a write action, or dynamodb:ReturnValues with a read action. DynamoDB provides dynamodb:Select only for read actions and dynamodb:ReturnValues only for write actions, so the condition here never takes effect and protects nothing. Pair each key with a supporting action, or remove it. This check applies to both Allow and Deny statements.

Security Warning – Untrustworthy condition key

Issue code: UNTRUSTWORTHY_CONDITION_KEY

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Untrustworthy condition key: The {{key}} condition key is not recommended for access
 control as it can be spoofed/manipulated by the caller.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{key}} condition key is not recommended for access control as
 it can be spoofed/manipulated by the caller."
```

Resolving the security warning

Do not use this condition key for access control. The caller can potentially manipulate or spoof the key value, which creates a security risk.

Related terms

- AWS global condition context keys

Security Warning – Allow with NotPrincipal

Issue code: ALLOW_WITH_NOT_PRINCIPAL

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Allow with NotPrincipal: Using Allow with NotPrincipal can be overly permissive. We
 recommend that you use Principal instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using Allow with NotPrincipal can be overly permissive. We recommend
 that you use Principal instead."
```

Resolving the security warning

Using `"Effect": "Allow"` with the `NotPrincipal` can be overly permissive. For example, this can grant permissions to anonymous principals. AWS recommends that you specify principals that need access using the `Principal` element. Alternatively, you can allow broad access and then add another statement that uses the `NotPrincipal` element with `“Effect”: “Deny”`.

- AWS JSON policy elements: Principal
- AWS JSON policy elements: NotPrincipal

Security Warning – ForAllValues with single valued key

Issue code: FORALLVALUES_WITH_SINGLE_VALUED_KEY

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
ForAllValues with single valued key: Using ForAllValues qualifier with the single-
valued condition key {{key}} can be overly permissive. We recommend that you remove
 ForAllValues:.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using ForAllValues qualifier with the single-valued condition key
 {{key}} can be overly permissive. We recommend that you remove ForAllValues:."
```

Resolving the security warning

AWS recommends that you use the `ForAllValues` only with multivalued conditions. The `ForAllValues` set operator tests whether the value of every member of the request set is a subset of the condition key set. The condition returns true if every key value in the request matches at least one value in the policy. It also returns true if there are no keys in the request, or if the key values resolve to a null data set, such as an empty string.

To learn whether a condition supports a single value or multiple values, review the Actions, resources, and condition keys page for the service. Condition keys with the `ArrayOf` data type

prefix are multivalued condition keys. For example, Amazon SES supports keys with single values (`String`) and the `ArrayOfString` multivalued data type.

- Multivalued context keys

Security Warning – Pass role with NotResource

Issue code: PASS_ROLE_WITH_NOT_RESOURCE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Pass role with NotResource: Using the iam:PassRole action with NotResource can be
 overly permissive because it can allow iam:PassRole permissions on multiple resources.
 We recommend that you specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the iam:PassRole action with NotResource can be overly
 permissive because it can allow iam:PassRole permissions on multiple resources. We
 recommend that you specify resource ARNs instead."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Using `iam:PassRole` in a policy with the `NotResource` element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- Passing a role to a service
- iam:PassedToService
- IAM JSON policy elements: NotResource
- IAM JSON policy elements: Resource

Security Warning – Pass role with star in action and NotResource

Issue code: PASS_ROLE_WITH_STAR_IN_ACTION_AND_NOT_RESOURCE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Pass role with star in action and NotResource: Using an action with a wildcard (*) and
 NotResource can be overly permissive because it can allow iam:PassRole permissions on
 multiple resources. We recommend that you specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using an action with a wildcard (*) and NotResource can be overly
 permissive because it can allow iam:PassRole permissions on multiple resources. We
 recommend that you specify resource ARNs instead."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Policies with a wildcard (*) in the `Action` and that include the `NotResource` element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- Passing a role to a service
- iam:PassedToService
- IAM JSON policy elements: NotResource
- IAM JSON policy elements: Resource

Security Warning – Pass role with NotAction and NotResource

Issue code: PASS_ROLE_WITH_NOT_ACTION_AND_NOT_RESOURCE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Pass role with NotAction and NotResource: Using NotAction with NotResource can
 be overly permissive because it can allow iam:PassRole permissions on multiple
 resources.. We recommend that you specify resource ARNs instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using NotAction with NotResource can be overly permissive because
 it can allow iam:PassRole permissions on multiple resources.. We recommend that you
 specify resource ARNs instead."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Using the `NotAction` element and listing some resources in the `NotResource` element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- Passing a role to a service
- iam:PassedToService
- IAM JSON policy elements: NotAction
- IAM JSON policy elements: Action
- IAM JSON policy elements: NotResource
- IAM JSON policy elements: Resource

Security Warning – Pass role with star in resource

Issue code: PASS_ROLE_WITH_STAR_IN_RESOURCE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Pass role with star in resource: Using the iam:PassRole action with wildcards (*)
 in the resource can be overly permissive because it allows iam:PassRole permissions
 on multiple resources. We recommend that you specify resource ARNs or add the
 iam:PassedToService condition key to your statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the iam:PassRole action with wildcards (*) in the resource can
 be overly permissive because it allows iam:PassRole permissions on multiple resources.
 We recommend that you specify resource ARNs or add the iam:PassedToService condition
 key to your statement."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Policies that allow `iam:PassRole` and that include a wildcard (*) in the `Resource` element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

Some AWS services include their service namespace in the name of their role. This policy check takes these conventions into account while analyzing the policy to generate findings. For example, the following resource ARN might not generate a finding:

```
arn:aws:iam::*:role/Service*
```

- Passing a role to a service
- iam:PassedToService
- IAM JSON policy elements: Resource

AWS managed policies with this security warning

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

One of those use cases is for administrators within your account. The following AWS managed policies provide administrator access and grant permissions to pass any IAM role to any service. AWS recommends that you attach the following AWS managed policies only to IAM identities that you consider administrators.

- AdministratorAccess-Amplify

The following AWS managed policies include permissions to `iam:PassRole` with a wildcard (*) in the resource and are on a deprecation path. For each of these policies, we updated the permission guidance, such as recommending a new AWS managed policy that supports the use case. To view alternatives to these policies, see the guides for each service.

- AWSElasticBeanstalkFullAccess
- AWSElasticBeanstalkService
- AWSLambdaFullAccess
- AWSLambdaReadOnlyAccess
- AWSOpsWorksFullAccess
- AWSOpsWorksRole
- AWSDataPipelineRole
- AmazonDynamoDBFullAccesswithDataPipeline
- AmazonElasticMapReduceFullAccess
- AmazonDynamoDBFullAccesswithDataPipeline
- AmazonEC2ContainerServiceFullAccess

The following AWS managed policies provide permissions for only service-linked roles, which allow AWS services to perform actions on your behalf. You cannot attach these policies to your IAM identities.

- AWSServiceRoleForAmazonEKSNodegroup

Security Warning – Pass role with star in action and resource

Issue code: PASS_ROLE_WITH_STAR_IN_ACTION_AND_RESOURCE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Pass role with star in action and resource: Using wildcards (*) in the action and the
 resource can be overly permissive because it allows iam:PassRole permissions on all
 resources. We recommend that you specify resource ARNs or add the iam:PassedToService
 condition key to your statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using wildcards (*) in the action and the resource can be overly
 permissive because it allows iam:PassRole permissions on all resources. We recommend
 that you specify resource ARNs or add the iam:PassedToService condition key to your
 statement."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Policies with a wildcard (*) in the `Action` and `Resource` elements can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- Passing a role to a service
- iam:PassedToService
- IAM JSON policy elements: Action
- IAM JSON policy elements: Resource

AWS managed policies with this security warning

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

Some of those use cases are for administrators within your account. The following AWS managed policies provide administrator access and grant permissions to pass any IAM role to any AWS

service. AWS recommends that you attach the following AWS managed policies to only the IAM identities that you consider administrators.

- AdministratorAccess
- IAMFullAccess

Security Warning – Pass role with star in resource and NotAction

Issue code: PASS_ROLE_WITH_STAR_IN_RESOURCE_AND_NOT_ACTION

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Pass role with star in resource and NotAction: Using a resource with wildcards (*) and
 NotAction can be overly permissive because it allows iam:PassRole permissions on all
 resources. We recommend that you specify resource ARNs or add the iam:PassedToService
 condition key to your statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using a resource with wildcards (*) and NotAction can be overly
 permissive because it allows iam:PassRole permissions on all resources. We recommend
 that you specify resource ARNs or add the iam:PassedToService condition key to your
 statement."
```

Resolving the security warning

To configure many AWS services, you must pass an IAM role to the service. To allow this you must grant the `iam:PassRole` permission to an identity (user, group of users, or role). Using the `NotAction` element in a policy with a wildcard (*) in the `Resource` element can allow your principals to access more services or features than you intended. AWS recommends that you specify allowed ARNs in the `Resource` element instead. Additionally, you can reduce permissions to a single service by using the `iam:PassedToService` condition key.

- Passing a role to a service

- iam:PassedToService
- IAM JSON policy elements: NotAction
- IAM JSON policy elements: Action
- IAM JSON policy elements: Resource

Security Warning – Missing paired condition keys

Issue code: MISSING_PAIRED_CONDITION_KEYS

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing paired condition keys: Using the condition key {{conditionKeyName}}
 can be overly permissive without also using the following condition keys:
 {{recommendedKeys}}. Condition keys like this one are more secure when paired with
 a related key. We recommend that you add the related condition keys to the same
 condition block.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the condition key {{conditionKeyName}} can be overly
 permissive without also using the following condition keys: {{recommendedKeys}}.
 Condition keys like this one are more secure when paired with a related key. We
 recommend that you add the related condition keys to the same condition block."
```

Resolving the security warning

Some condition keys are more secure when paired with other related condition keys. AWS recommends that you include the related condition keys in the same condition block as the existing condition key. This makes the permissions granted through the policy more secure.

For example, you can use the `aws:VpcSourceIp` condition key to compare the IP address from which a request was made with the IP address that you specify in the policy. AWS recommends that you add the related `aws:SourceVPC` condition key. This checks whether the request comes from the VPC that you specify in the policy *and* the IP address that you specify.

Related terms

- aws:VpcSourceIp global condition key
- aws:SourceVPC global condition key
- Global condition keys
- Condition element
- Overview of JSON policies

Security Warning – Deny with unsupported tag condition key for service

Issue code: DENY_WITH_UNSUPPORTED_TAG_CONDITION_KEY_FOR_SERVICE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Deny with unsupported tag condition key for service: Using the effect Deny with the
 tag condition key {{conditionKeyName}} and actions for services with the following
 prefixes can be overly permissive: {{serviceNames}}. Actions for the listed services
 are not denied by this statement. We recommend that you move these actions to a
 different statement without this condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the effect Deny with the tag condition key
 {{conditionKeyName}} and actions for services with the following prefixes can be
 overly permissive: {{serviceNames}}. Actions for the listed services are not denied
 by this statement. We recommend that you move these actions to a different statement
 without this condition key."
```

Resolving the security warning

Using unsupported tag condition keys in the `Condition` element of a policy with `"Effect": "Deny"` can be overly permissive, because the condition is ignored for that service. AWS recommends that you remove the service actions that don't support the condition key and create another statement to deny access to specific resources for those actions.

If you use the `aws:ResourceTag` condition key and it's not supported by a service action, then the key is not included in the request context. In this case, the condition in the `Deny` statement always returns `false` and the action is never denied. This happens even if the resource is tagged correctly.

When a service supports the `aws:ResourceTag` condition key, you can use tags to control access to that service's resources. This is known as attribute-based access control (ABAC). Services that don't support these keys require you to control access to resources using resource-based access control (RBAC).

> **Note**
>
> Some services allow support for the `aws:ResourceTag` condition key for a subset of their resources and actions. IAM Access Analyzer returns findings for the service actions that are not supported. For example, Amazon S3 supports `aws:ResourceTag` for a subset of its resources. To view all of the resource types available in Amazon S3 that support the `aws:ResourceTag` condition key, see Resource types defined by Amazon S3 in the Service Authorization Reference.

For example, assume that you want to deny access to untag delete specific resources that are tagged with the key-value pair `status=Confidential`. Also assume that AWS Lambda allows you to tag and untag resources, but doesn't support the `aws:ResourceTag` condition key. To deny the delete actions for AWS App Mesh and AWS Backup if this tag is present, use the `aws:ResourceTag` condition key. For Lambda, use a resource naming convention that includes the `"Confidential"` prefix. Then include a separate statement that prevents deleting resources with that naming convention.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyDeleteSupported",
            "Effect": "Deny",
            "Action": [
                "appmesh:DeleteMesh",
                "backup:DeleteBackupPlan"
                ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:ResourceTag/status": "Confidential"
                }
            }
        },
        {
            "Sid": "DenyDeleteUnsupported",
            "Effect": "Deny",
            "Action": "lambda:DeleteFunction",
            "Resource": "arn:aws:lambda:*:123456789012:function:status-Confidential*"
        }
    ]
}
```

Warning

Do not use the …IfExists version of the condition operator as a workaround for this finding. This means "Deny the action if the key is present in the request context and the values match. Otherwise, deny the action." In the previous example, including the `lambda:DeleteFunction` action in the `DenyDeleteSupported` statement with the `StringEqualsIfExists` operator always denies the action. For that action, the key is not present in the context, and every attempt to delete that resource type is denied, regardless of whether the resource is tagged.

Related terms

- Global condition keys
- Comparing ABAC to RBAC
- IAM JSON policy elements: Condition operators
- Condition element
- Overview of JSON policies

Security Warning – Deny NotAction with unsupported tag condition key for service

Issue code: DENY_NOTACTION_WITH_UNSUPPORTED_TAG_CONDITION_KEY_FOR_SERVICE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Deny NotAction with unsupported tag condition key for service: Using the effect Deny
 with NotAction and the tag condition key {{conditionKeyName}} can be overly permissive
 because some service actions are not denied by this statement. This is because the
 condition key doesn't apply to some service actions. We recommend that you use Action
 instead of NotAction.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the effect Deny with NotAction and the tag condition key
 {{conditionKeyName}} can be overly permissive because some service actions are not
 denied by this statement. This is because the condition key doesn't apply to some
 service actions. We recommend that you use Action instead of NotAction."
```

Resolving the security warning

Using tag condition keys in the `Condition` element of a policy with the element `NotAction` and `"Effect": "Deny"` can be overly permissive. The condition is ignored for service actions that don't support the condition key. AWS recommends that you rewrite the logic to deny a list of actions.

If you use the `aws:ResourceTag` condition key with `NotAction`, any new or existing service actions that don't support the key are not denied. AWS recommends that you explicitly list the actions that you want to deny. IAM Access Analyzer returns a separate finding for listed actions that don't support the `aws:ResourceTag` condition key. For more information, see Security Warning – Deny with unsupported tag condition key for service.

When a service supports the `aws:ResourceTag` condition key, you can use tags to control access to that service's resources. This is known as attribute-based access control (ABAC). Services that don't support these keys require you to control access to resources using resource-based access control (RBAC).

Related terms

- Global condition keys
- Comparing ABAC to RBAC

- IAM JSON policy elements: Condition operators
- Condition element
- Overview of JSON policies

Security Warning – Restrict access to service principal

Issue code: RESTRICT_ACCESS_TO_SERVICE_PRINCIPAL

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Restrict access to service principal: Granting access to a service principal without
 specifying a source is overly permissive. Use aws:SourceArn, aws:SourceAccount,
 aws:SourceOrgID, or aws:SourceOrgPaths condition key to grant fine-grained access.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Granting access to a service principal without specifying a source
 is overly permissive. Use aws:SourceArn, aws:SourceAccount, aws:SourceOrgID, or
 aws:SourceOrgPaths condition key to grant fine-grained access."
```

Resolving the security warning

You can specify AWS services in the `Principal` element of a resource-based policy using a service principal, which is an identifier for the service. When granting access to a service principal to act on your behalf, restrict access. You can prevent overly permissive policies by using the `aws:SourceArn`, `aws:SourceAccount`, `aws:SourceOrgID`, or `aws:SourceOrgPaths` condition keys to restrict access to a specific source, such as a specific resource ARN, AWS account, organization ID, or organization paths. Restricting access helps you prevent a security issue called *the confused deputy problem*.

Related terms

- AWS service principals
- AWS global condition keys: aws:SourceAccount

- AWS global condition keys: aws:SourceArn
- AWS global condition keys: aws:SourceOrgId
- AWS global condition keys: aws:SourceOrgPaths
- The confused deputy problem

Security Warning – Missing condition key for oidc principal

Issue code: MISSING_CONDITION_KEY_FOR_OIDC_PRINCIPAL

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing condition key for oidc principal: Using an Open ID Connect principal without a
 condition can be overly permissive. Add condition keys with a prefix that matches your
 federated OIDC principals to ensure that only the intended identity provider assumes
 the role.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using an Open ID Connect principal without a condition can be
 overly permissive. Add condition keys with a prefix that matches your federated OIDC
 principals to ensure that only the intended identity provider assumes the role."
```

Resolving the security warning

Using an Open ID Connect principal without a condition can be overly permissive. Add condition keys with a prefix that matches your federated OIDC principals to ensure that only the intended identity provider assumes the role.

Related terms

- Creating a role for web identity or OpenID Connect Federation (console)

Security Warning – Missing github repo condition key

Issue code: MISSING_GITHUB_REPO_CONDITION_KEY

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Missing github repo condition key: Granting a federated GitHub principal permissions
 without a condition key can allow more sources to assume the role than you intended.
 Add the token.actions.githubusercontent.com:sub condition key and specify the branch
 and repository name in the value.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Granting a federated GitHub principal permissions without a
 condition key can allow more sources to assume the role than you intended. Add the
 token.actions.githubusercontent.com:sub condition key and specify the branch and
 repository name in the value."
```

Resolving the security warning

If you use GitHub as an OIDC IdP, best practice is to limit the entities that can assume the role associated with the IAM IdP. When you include a `Condition` statement in a role trust policy, you can limit the role to a specific GitHub organization, repository, or branch. You can use the condition key `token.actions.githubusercontent.com:sub` to limit access. We recommend that you limit the condition to a specific set of repositories or branches. If you do not include this condition, then GitHub Actions from organizations or repositories outside of your control are able to assume roles associated with the GitHub IAM IdP in your AWS account.

Related terms

- Configuring a role for GitHub OIDC identity provider

Security Warning – String like operator with ARN condition keys

Issue code: STRING_LIKE_OPERATOR_WITH_ARN_CONDITION_KEYS

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
String like operator with ARN condition keys: Use the operator type {{allowed}} instead
 of operator {{operator}} for the condition key {{key}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Use the operator type {{allowed}} instead of operator {{operator}}
 for the condition key {{key}}."
```

Resolving the security warning

AWS recommends that you use ARN operators instead of string operators when comparing ARNs to ensure proper access restriction based on ARN condition values. Update the `StringLike` operator to the `ArnLike` operator in your `Condition` element whenever the specified key is used.

These AWS managed policies are exceptions to this security warning:

- AmazonSecurityLakeAdministrator
- AWSCodePipeline_FullAccess
- AWSCodePipeline_ReadOnlyAccess
- S3UnlockBucketPolicy
- SQSUnlockQueuePolicy

Related terms

- Amazon Resource Name (ARN) condition operators
- String condition operators
- AWS managed policies

Security Warning – DynamoDB attributes without select

Issue code: DYNAMODB_ATTRIBUTES_WITHOUT_SELECT

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB attributes without select: Restricting dynamodb:Attributes without also
 setting dynamodb:Select to SPECIFIC_ATTRIBUTES allows all attributes to be returned
 on read requests that omit a projection expression. We recommend that you also set
 dynamodb:Select to SPECIFIC_ATTRIBUTES.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Restricting dynamodb:Attributes without also setting dynamodb:Select
 to SPECIFIC_ATTRIBUTES allows all attributes to be returned on read requests that
 omit a projection expression. We recommend that you also set dynamodb:Select to
 SPECIFIC_ATTRIBUTES."
```

Resolving the security warning

Add a `StringEquals` condition that sets `dynamodb:Select` to `SPECIFIC_ATTRIBUTES` in the same statement as the `dynamodb:Attributes` condition. DynamoDB evaluates `dynamodb:Attributes` only for requests that specify the attributes to return. Without this condition, a read request that omits a projection expression returns the entire item.

Related terms

- Using IAM policy conditions for fine-grained access control
- IAM JSON policy elements: Condition operators

Security Warning – S3 prefix in negative context

Issue code: S3_PREFIX_IN_NEGATIVE_CONTEXT

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
S3 prefix in negative context: Using the s3:prefix condition key in a Deny statement or
 with a negated string operator does not block listing the objects under the matching
 prefixes. A request that specifies a shorter prefix does not match the condition and
```

```
can still list those objects. To restrict listing to specific prefixes, use s3:prefix
with StringLike in an Allow statement, or use a Deny statement with StringNotLike.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the s3:prefix condition key in a Deny statement or with
 a negated string operator does not block listing the objects under the matching
 prefixes. A request that specifies a shorter prefix does not match the condition and
 can still list those objects. To restrict listing to specific prefixes, use s3:prefix
 with StringLike in an Allow statement, or use a Deny statement with StringNotLike."
```

Resolving the security warning

To restrict listing to specific prefixes, use one of the following patterns:

- **Allow with StringLike** – Grant `s3:ListBucket` with a `StringLike` condition on `s3:prefix` listing the prefixes you want to permit. Requests that supply a different prefix are not covered by this Allow and cannot list those objects.
- **Deny with StringNotLike** – Deny `s3:ListBucket` with a `StringNotLike` condition on `s3:prefix` listing the prefixes you want to allow. A request that omits the prefix or supplies a shorter prefix does not match the allowed set and is denied.

A `Deny` with `StringLike` or an `Allow` with `StringNotLike` does not effectively restrict access because a caller can bypass the condition by specifying a shorter prefix (for example, `priv` instead of `private/`) that does not match the condition value.

Related terms

- Amazon S3 condition key examples
- IAM JSON policy elements: Condition operators
- IAM policy elements: Condition

Security Warning – ForAnyValue with audience claim type

Issue code: FORANYVALUE_WITH_AUDIENCE_CLAIM_TYPE

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
ForAnyValue with audience claim type: Using ForAnyValue qualifier with the single-
valued condition key {{key}} can be overly permissive. We recommend that you remove
 ForAnyValue:.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using ForAnyValue qualifier with the single-valued condition key
 {{key}} can be overly permissive. We recommend that you remove ForAnyValue:."
```

Resolving the security warning

AWS recommends that you do not use the `ForAnyValue` set operator with single-valued condition keys. Use set operators only with multivalued condition keys. Remove the `ForAnyValue` set operator.

Related terms

- Single-valued vs. multivalued context keys
- Single-valued context key policy examples

Security Warning – DynamoDB attributes without return values

Issue code: DYNAMODB_ATTRIBUTES_WITHOUT_RETURN_VALUES

Finding type: SECURITY_WARNING

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
DynamoDB attributes without return values: Restricting dynamodb:Attributes without
 limiting dynamodb:ReturnValues allows write operations to return the entire item,
 which can expose attributes outside the allowed list. We recommend that you restrict
 dynamodb:ReturnValues to NONE, UPDATED_OLD, or UPDATED_NEW.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Restricting dynamodb:Attributes without limiting
 dynamodb:ReturnValues allows write operations to return the entire item, which
 can expose attributes outside the allowed list. We recommend that you restrict
 dynamodb:ReturnValues to NONE, UPDATED_OLD, or UPDATED_NEW."
```

Resolving the security warning

Add a condition that limits `dynamodb:ReturnValues` to `NONE`, `UPDATED_OLD`, or `UPDATED_NEW` in the same statement as the `dynamodb:Attributes` condition. Without this limit, a write that sets `dynamodb:ReturnValues` to `ALL_OLD` or `ALL_NEW` returns the entire item, including attributes outside the ones you allowed.

Suggestion – Empty array action

Issue code: EMPTY_ARRAY_ACTION

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty array action: This statement includes no actions and does not affect the policy.
 Specify actions.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "This statement includes no actions and does not affect the policy.
 Specify actions."
```

Resolving the suggestion

Statements must include either an `Action` or `NotAction` element that includes a set of actions. When the element is empty, the policy statement provides no permissions. Specify actions in the `Action` element.

- IAM JSON policy elements: Action

Suggestion – Empty array condition

Issue code: EMPTY_ARRAY_CONDITION

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty array condition: There are no values for the condition key {{key}} and it does
 not affect the policy. Specify conditions.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "There are no values for the condition key {{key}} and it does not
 affect the policy. Specify conditions."
```

Resolving the suggestion

The optional `Condition` element structure requires that you use a condition operator and a key-value pair. When the condition value is empty, the condition returns `true` and the policy statement provides no permissions. Specify a condition value.

- IAM JSON policy elements: Condition

Suggestion – Empty array condition ForAllValues

Issue code: EMPTY_ARRAY_CONDITION_FORALLVALUES

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty array condition ForAllValues: The ForAllValues prefix with an empty condition key
 matches only if the key {{key}} is missing from the request context. To determine if
 the request context is empty, we recommend that you use the Null condition operator
 with the value of true instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The ForAllValues prefix with an empty condition key matches only
 if the key {{key}} is missing from the request context. To determine if the request
 context is empty, we recommend that you use the Null condition operator with the value
 of true instead."
```

Resolving the suggestion

The `Condition` element structure requires that you use a condition operator and a key-value pair. The `ForAllValues` set operator tests whether the value of every member of the request set is a subset of the condition key set.

When you use `ForAllValues` with an empty condition key, the condition matches only if there are no keys in the request. AWS recommends that if you want to test whether a request context is empty, use the `Null` condition operator instead.

- Multivalued context keys
- Null condition operator
- IAM JSON policy elements: Condition

Suggestion – Empty array condition ForAnyValue

Issue code: EMPTY_ARRAY_CONDITION_FORANYVALUE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty array condition ForAnyValue: The ForAnyValue prefix with an empty condition key
 {{key}} never matches the request context and it does not affect the policy. Specify
 conditions.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The ForAnyValue prefix with an empty condition key {{key}} never
 matches the request context and it does not affect the policy. Specify conditions."
```

Resolving the suggestion

The `Condition` element structure requires that you use a condition operator and a key-value pair. The `ForAnyValues` set operator tests whether at least one member of the set of request values matches at least one member of the set of condition key values.

When you use `ForAnyValues` with an empty condition key, the condition never matches. This means that the statement has no effect on the policy. AWS recommends that you rewrite the condition.

- Multivalued context keys
- IAM JSON policy elements: Condition

Suggestion – Empty array condition IfExists

Issue code: EMPTY_ARRAY_CONDITION_IFEXISTS

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty array condition IfExists: The IfExists suffix with an empty condition key matches
 only if the key {{key}} is missing from the request context. To determine if the
 request context is empty, we recommend that you use the Null condition operator with
 the value of true instead.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The IfExists suffix with an empty condition key matches only if the
 key {{key}} is missing from the request context. To determine if the request context
 is empty, we recommend that you use the Null condition operator with the value of true
 instead."
```

Resolving the suggestion

The `...IfExists` suffix edits a condition operator. It means that if the policy key is present in the context of the request, process the key as specified in the policy. If the key is not present, evaluate the condition element as true.

When you use `...IfExists` with an empty condition key, the condition matches only if there are no keys in the request. AWS recommends that if you want to test whether a request context is empty, use the `Null` condition operator instead.

- ...IfExists condition operators
- IAM JSON policy elements: Condition

Suggestion – Empty array principal

Issue code: EMPTY_ARRAY_PRINCIPAL

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty array principal: This statement includes no principals and does not affect the
 policy. Specify principals.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "This statement includes no principals and does not affect the
 policy. Specify principals."
```

Resolving the suggestion

You must use the `Principal` or `NotPrincipal` element in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in a resource.

When you provide an empty array in a statement's `Principal` element, the statement has no effect on the policy. AWS recommends that you specify the principals that should have access to the resource.

- IAM JSON policy elements: Principal

- IAM JSON policy elements: NotPrincipal

Suggestion – Empty array resource

Issue code: EMPTY_ARRAY_RESOURCE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty array resource: This statement includes no resources and does not affect the
 policy. Specify resources.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "This statement includes no resources and does not affect the policy.
 Specify resources."
```

Resolving the suggestion

Statements must include either a `Resource` or a `NotResource` element.

When you provide an empty array in a statement's resource element, the statement has no effect on the policy. AWS recommends that you specify Amazon Resource Names (ARNs) for resources.

- IAM JSON policy elements: Resource
- IAM JSON policy elements: NotResource

Suggestion – Empty object condition

Issue code: EMPTY_OBJECT_CONDITION

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty object condition: This condition block is empty and it does not affect the
 policy. Specify conditions.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "This condition block is empty and it does not affect the policy.
 Specify conditions."
```

Resolving the suggestion

The `Condition` element structure requires that you use a condition operator and a key-value pair.

When you provide an empty object in a statement's condition element, the statement has no effect on the policy. Remove the optional element or specify conditions.

- IAM JSON policy elements: Condition

Suggestion – Empty object principal

Issue code: EMPTY_OBJECT_PRINCIPAL

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty object principal: This statement includes no principals and does not affect the
 policy. Specify principals.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "This statement includes no principals and does not affect the
 policy. Specify principals."
```

Resolving the suggestion

You must use the `Principal` or `NotPrincipal` element in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in a resource.

When you provide an empty object in a statement's `Principal` element, the statement has no effect on the policy. AWS recommends that you specify the principals that should have access to the resource.

- IAM JSON policy elements: Principal
- IAM JSON policy elements: NotPrincipal

Suggestion – Empty Sid value

Issue code: EMPTY_SID_VALUE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Empty Sid value: Add a value to the empty string in the Sid element.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Add a value to the empty string in the Sid element."
```

Resolving the suggestion

The optional `Sid` (statement ID) element allows you to enter an identifier that you provide for the policy statement. You can assign an `Sid` value to each statement in a statement array. If you choose to use the `Sid` element, you must provide a string value.

Related terms

- IAM JSON policy elements: Sid

Suggestion – Equivalent to null false

Issue code: EQUIVALENT_TO_NULL_FALSE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Equivalent to null false: We recommend replacing the key {{key}} in the condition block
 of {{operator}} with {{{recommendedKey}}: false} to ensure better enforcement of the
 condition.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "We recommend replacing the key {{key}} in the condition block of
 {{operator}} with {{{recommendedKey}}: false} to ensure better enforcement of the
 condition."
```

Resolving the suggestion

Replace the current condition key with the recommended key set to `false`. This change improves policy clarity and ensures more reliable condition evaluation. Update your condition block to use `{recommendedKey}: false` instead of the current key-operator combination.

Related terms

- IAM policy elements: Condition
- Conditions with multiple context keys or values
- AWS global condition context keys

Suggestion – Equivalent to null true

Issue code: EQUIVALENT_TO_NULL_TRUE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Equivalent to null true: We recommend replacing the key {{key}} in the condition block
 of {{operator}} with {{{recommendedKey}}: true} to ensure better enforcement of the
 condition.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "We recommend replacing the key {{key}} in the condition block of
 {{operator}} with {{{recommendedKey}}: true} to ensure better enforcement of the
 condition."
```

Resolving the suggestion

Replace the current condition key with the recommended key set to `true`. This change improves policy clarity and ensures more reliable condition evaluation. Update your condition block to use `{recommendedKey}: true` instead of the current key-operator combination.

Related terms

- IAM policy elements: Condition
- Conditions with multiple context keys or values
- AWS global condition context keys

Suggestion – Improve IP range

Issue code: IMPROVE_IP_RANGE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Improve IP range: The non-zero bits in the IP address after the masked bits are
 ignored. Replace address with {{addr}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The non-zero bits in the IP address after the masked bits are
 ignored. Replace address with {{addr}}."
```

Resolving the suggestion

IP address conditions must be in the standard CIDR format, such as 203.0.113.0/24 or 2001:DB8:1234:5678::/64. When you include non-zero bits after the masked bits, they are not considered for the condition. AWS recommends that you use the new address included in the message.

- IP address condition operators
- IAM JSON policy elements: Condition

Suggestion – Null with qualifier

Issue code: NULL_WITH_QUALIFIER

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Null with qualifier: Avoid using the Null condition operator with the ForAllValues or
 ForAnyValue qualifiers because they always return a true or false respectively.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Avoid using the Null condition operator with the ForAllValues or
 ForAnyValue qualifiers because they always return a true or false respectively."
```

Resolving the suggestion

In the `Condition` element, you build expressions in which you use condition operators like equal or less than to compare a condition in the policy against keys and values in the request context. For requests that include multiple values for a single condition key, you must use the `ForAllValues` or `ForAnyValue` set operators.

When you use the `Null` condition operator with `ForAllValues`, the statement always returns `true`. When you use the `Null` condition operator with `ForAnyValue`, the statement always returns `false`. AWS recommends that you use the `StringLike` condition operator with these set operators.

Related terms

- Multivalued context keys
- Null condition operator
- Condition element

Suggestion – Private IP address subset

Issue code: PRIVATE_IP_ADDRESS_SUBSET

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Private IP address subset: The values for condition key aws:SourceIp include a mix
 of private and public IP addresses. The private addresses will not have the desired
 effect. aws:SourceIp works only for public IP address ranges. To define permissions
 for private IP ranges, use aws:VpcSourceIp.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The values for condition key aws:SourceIp include a mix of private
 and public IP addresses. The private addresses will not have the desired effect.
 aws:SourceIp works only for public IP address ranges. To define permissions for
 private IP ranges, use aws:VpcSourceIp."
```

Resolving the suggestion

The global condition key `aws:SourceIp` works only for public IP address ranges.

When your `Condition` element includes a mix of private and public IP addresses, the statement might not have the desired effect. You can specify private IP addresses using `aws:VpcSourceIP`.

> **Note**
>
> The global condition key `aws:VpcSourceIP` matches only if the request originates from the specified IP address and it goes through a VPC endpoint.

- aws:SourceIp global condition key
- aws:VpcSourceIp global condition key
- IP address condition operators
- IAM JSON policy elements: Condition

Suggestion – Private NotIpAddress subset

Issue code: PRIVATE_NOT_IP_ADDRESS_SUBSET

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Private NotIpAddress subset: The values for condition key aws:SourceIp include a mix of
 private and public IP addresses. The private addresses have no effect. aws:SourceIp
 works only for public IP address ranges. To define permissions for private IP ranges,
 use aws:VpcSourceIp.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The values for condition key aws:SourceIp include a mix of private
 and public IP addresses. The private addresses have no effect. aws:SourceIp works
 only for public IP address ranges. To define permissions for private IP ranges, use
 aws:VpcSourceIp."
```

Resolving the suggestion

The global condition key `aws:SourceIp` works only for public IP address ranges.

When your `Condition` element includes the `NotIpAddress` condition operator and a mix of private and public IP addresses, the statement might not have the desired effect. Every public IP addresses that is not specified in the policy will match. No private IP addresses will match. To achieve this effect, you can use `NotIpAddress` with `aws:VpcSourceIP` and specify the private IP addresses that should not match.

- aws:SourceIp global condition key
- aws:VpcSourceIp global condition key

- IP address condition operators
- IAM JSON policy elements: Condition

Suggestion – Redundant action

Issue code: REDUNDANT_ACTION

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant action: The {{redundantActionCount}} action(s) are redundant because they
 provide similar permissions. Update the policy to remove the redundant action such as:
 {{redundantAction}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{redundantActionCount}} action(s) are redundant because they
 provide similar permissions. Update the policy to remove the redundant action such as:
 {{redundantAction}}."
```

Resolving the suggestion

When you use wildcards (*) in the `Action` element, you can include redundant permissions. AWS recommends that you review your policy and include only the permissions that you need. This can help you remove redundant actions.

For example, the following actions include the `iam:GetCredentialReport` action twice.

```
"Action": [
        "iam:Get*",
        "iam:List*",
        "iam:GetCredentialReport"
    ],
```

In this example, permissions are defined for every IAM action that begins with `Get` or `List`. When IAM adds additional get or list operations, this policy will allow them. You might want to allow all

of these read-only actions. The `iam:GetCredentialReport` action is already included as part of `iam:Get*`. To remove the duplicate permissions, you could remove `iam:GetCredentialReport`.

You receive a finding for this policy check when all of the contents of an action are redundant. In this example, if the element included `iam:*CredentialReport`, it is not considered redundant. That includes `iam:GetCredentialReport`, which is redundant, and `iam:GenerateCredentialReport`, which is not. Removing either `iam:Get*` or `iam:*CredentialReport` would change the policy's permissions.

- IAM JSON policy elements: Action

AWS managed policies with this suggestion

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

Redundant actions do not affect the permissions granted by the policy. When using an AWS managed policy as a reference to create your customer managed policy, AWS recommends that you remove redundant actions from your policy.

Suggestion – Redundant condition value num

Issue code: REDUNDANT_CONDITION_VALUE_NUM

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant condition value num: Multiple values in {{operator}} are redundant. Replace
 with the {{greatest/least}} single value for {{key}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Multiple values in {{operator}} are redundant. Replace with the
 {{greatest/least}} single value for {{key}}."
```

Resolving the suggestion

When you use numeric condition operators for similar values in a condition key, you can create an overlap that results in redundant permissions.

For example, the following `Condition` element includes multiple `aws:MultiFactorAuthAge` conditions that have an age overlap of 1200 seconds.

```
"Condition": {
        "NumericLessThan": {
          "aws:MultiFactorAuthAge": [
            "2700",
            "3600"
          ]
        }
      }
```

In this example, the permissions are defined if multi-factor authentication (MFA) was completed less than 3600 seconds (1 hour) ago. You could remove the redundant `2700` value.

- Numeric condition operators
- IAM JSON policy elements: Condition

Suggestion – Redundant resource

Issue code: REDUNDANT_RESOURCE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant resource: The {{redundantResourceCount}} resource ARN(s) are redundant
 because they reference the same resource. Review the use of wildcards (*)
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The {{redundantResourceCount}} resource ARN(s) are redundant because
 they reference the same resource. Review the use of wildcards (*)"
```

Resolving the suggestion

When you use wildcards (*) in Amazon Resource Names (ARNs), you can create redundant resource permissions.

For example, the following `Resource` element includes multiple ARNs with redundant permissions.

```
"Resource": [
            "arn:aws:iam::111122223333:role/jane-admin",
            "arn:aws:iam::111122223333:role/jane-s3only",
            "arn:aws:iam::111122223333:role/jane*"
        ],
```

In this example, the permissions are defined for any role with a name starting with `jane`. You could remove the redundant `jane-admin` and `jane-s3only` ARNs without changing the resulting permissions. This does make the policy dynamic. It will define permissions for any future roles that begin with `jane`. If the intention of the policy is to allow access to a static number of roles, then remove the last ARN and list only the ARNs that should be defined.

- IAM JSON policy elements: Resource

AWS managed policies with this suggestion

AWS managed policies enable you to get started with AWS by assigning permissions based on general AWS use cases.

Redundant resources do not affect the permissions granted by the policy. When using an AWS managed policy as a reference to create your customer managed policy, AWS recommends that you remove redundant resources from your policy.

Suggestion – Redundant statement

Issue code: REDUNDANT_STATEMENT

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant statement: The statements are redundant because they provide identical
 permissions. Update the policy to remove the redundant statement.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The statements are redundant because they provide identical
 permissions. Update the policy to remove the redundant statement."
```

Resolving the suggestion

The `Statement` element is the main element for a policy. This element is required. The `Statement` element can contain a single statement or an array of individual statements.

When you include the same statement more than once in a long policy, the statements are is redundant. You can remove one of the statements without affecting the permissions granted by the policy. When someone edits a policy, they might change one of the statements without updating the duplicate. This might result in more permissions than intended.

- IAM JSON policy elements: Statement

Suggestion – Wildcard in service name

Issue code: WILDCARD_IN_SERVICE_NAME

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Wildcard in service name: Avoid using wildcards (*, ?) in the service name because it
 might grant unintended access to other AWS services with similar names.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Avoid using wildcards (*, ?) in the service name because it might
 grant unintended access to other AWS services with similar names."
```

Resolving the suggestion

When you include the name of an AWS service in a policy, AWS recommends that you do not include wildcards (*, ?). This might add permissions for future services that you do not intend. For example, there are more than a dozen AWS services with the word `*code*` in their name.

```
"Resource": "arn:aws:*code*::111122223333:*"
```

- IAM JSON policy elements: Resource

Suggestion – Allow with unsupported tag condition key for service

Issue code: ALLOW_WITH_UNSUPPORTED_TAG_CONDITION_KEY_FOR_SERVICE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Allow with unsupported tag condition key for service: Using the effect Allow with the
 tag condition key {{conditionKeyName}} and actions for services with the following
 prefixes does not affect the policy: {{serviceNames}}. Actions for the listed service
 are not allowed by this statement. We recommend that you move these actions to a
 different statement without this condition key.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the effect Allow with the tag condition key
 {{conditionKeyName}} and actions for services with the following prefixes does not
 affect the policy: {{serviceNames}}. Actions for the listed service are not allowed
 by this statement. We recommend that you move these actions to a different statement
 without this condition key."
```

Resolving the suggestion

Using unsupported tag condition keys in the `Condition` element of a policy with `"Effect": "Allow"` does not affect the permissions granted by the policy, because the condition is ignored

for that service action. AWS recommends that you remove the actions for services that don't support the condition key and create another statement to allow access to specific resources in that service.

If you use the `aws:ResourceTag` condition key and it's not supported by a service action, then the key is not included in the request context. In this case, the condition in the `Allow` statement always returns `false` and the action is never allowed. This happens even if the resource is tagged correctly.

When a service supports the `aws:ResourceTag` condition key, you can use tags to control access to that service's resources. This is known as attribute-based access control (ABAC). Services that don't support these keys require you to control access to resources using resource-based access control (RBAC).

> **Note**
>
> Some services allow support for the `aws:ResourceTag` condition key for a subset of their resources and actions. IAM Access Analyzer returns findings for the service actions that are not supported. For example, Amazon S3 supports `aws:ResourceTag` for a subset of its resources. To view all of the resource types available in Amazon S3 that support the `aws:ResourceTag` condition key, see Resource types defined by Amazon S3 in the Service Authorization Reference.

For example, assume that you want to allow team members to view details for specific resources that are tagged with the key-value pair `team=BumbleBee`. Also assume that AWS Lambda allows you to tag resources, but doesn't support the `aws:ResourceTag` condition key. To allow view actions for AWS App Mesh and AWS Backup if this tag is present, use the `aws:ResourceTag` condition key. For Lambda, use a resource naming convention that includes the team name as a prefix. Then include a separate statement that allows viewing resources with that naming convention.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowViewSupported",
            "Effect": "Allow",
            "Action": [
```

```
                "appmesh:DescribeMesh",
                "backup:GetBackupPlan"
                ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/team": "BumbleBee"
                }
            }
        },
        {
            "Sid": "AllowViewUnsupported",
            "Effect": "Allow",
            "Action": "lambda:GetFunction",
            "Resource": "arn:aws:lambda:*:123456789012:function:team-BumbleBee*"
        }
    ]
}
```

Warning

Do not use the `Not` version of the condition operator with `"Effect": "Allow"` as a workaround for this finding. These condition operators provide negated matching. This means that after the condition is evaluated, the result is negated. In the previous example, including the `lambda:GetFunction` action in the `AllowViewSupported` statement with the `StringNotEquals` operator always allows the action, regardless of whether the resource is tagged.

Do not use the `...IfExists` version of the condition operator as a workaround for this finding. This means "Allow the action if the key is present in the request context and the values match. Otherwise, allow the action." In the previous example, including the `lambda:GetFunction` action in the `AllowViewSupported` statement with the `StringEqualsIfExists` operator always allows the action. For that action, the key is not present in the context, and every attempt to view that resource type is allowed, regardless of whether the resource is tagged.

Related terms

- Global condition keys
- IAM JSON policy elements: Condition operators

- Condition element
- Overview of JSON policies

Suggestion – Allow NotAction with unsupported tag condition key for service

Issue code: ALLOW_NOTACTION_WITH_UNSUPPORTED_TAG_CONDITION_KEY_FOR_SERVICE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Allow NotAction with unsupported tag condition key for service: Using the effect Allow
 with NotAction and the tag condition key {{conditionKeyName}} allows only service
 actions that support the condition key. The condition key doesn't apply to some
 service actions. We recommend that you use Action instead of NotAction.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "Using the effect Allow with NotAction and the tag condition key
 {{conditionKeyName}} allows only service actions that support the condition key. The
 condition key doesn't apply to some service actions. We recommend that you use Action
 instead of NotAction."
```

Resolving the suggestion

Using unsupported tag condition keys in the `Condition` element of a policy with the element `NotAction` and `"Effect": "Allow"` does not affect the permissions granted by the policy. The condition is ignored for service actions that don't support the condition key. AWS recommends that you rewrite the logic to allow a list of actions.

If you use the `aws:ResourceTag` condition key with `NotAction`, any new or existing service actions that don't support the key are not allowed. AWS recommends that you explicitly list the actions that you want to allow. IAM Access Analyzer returns a separate finding for listed actions that don't support the `aws:ResourceTag` condition key. For more information, see Suggestion – Allow with unsupported tag condition key for service.

When a service supports the `aws:ResourceTag` condition key, you can use tags to control access to that service's resources. This is known as attribute-based access control (ABAC). Services that

don't support these keys require you to control access to resources using resource-based access control (RBAC).

Related terms

- Global condition keys
- Comparing ABAC to RBAC
- IAM JSON policy elements: Condition operators
- Condition element
- Overview of JSON policies

Suggestion – Recommended condition key for service principal

Issue code: RECOMMENDED_CONDITION_KEY_FOR_SERVICE_PRINCIPAL

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Recommended condition key for service principal: To restrict access to the service
 principal {{servicePrincipalPrefix}} operating on your behalf, we recommend
 aws:SourceArn, aws:SourceAccount, aws:SourceOrgID, or aws:SourceOrgPaths instead of
 {{key}}.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "To restrict access to the service principal
 {{servicePrincipalPrefix}} operating on your behalf, we recommend aws:SourceArn,
 aws:SourceAccount, aws:SourceOrgID, or aws:SourceOrgPaths instead of {{key}}."
```

Resolving the suggestion

You can specify AWS services in the `Principal` element of a resource-based policy using a *service principal*, which is an identifier for the service. You should use the `aws:SourceArn`, `aws:SourceAccount`, `aws:SourceOrgID`, or `aws:SourceOrgPaths` condition keys when

granting access to service principals instead of other condition keys, such as `aws:Referer`. This helps you prevent a security issue called *the confused deputy problem*.

Related terms

- AWS service principals
- AWS global condition keys: aws:SourceAccount
- AWS global condition keys: aws:SourceArn
- AWS global condition keys: aws:SourceOrgId
- AWS global condition keys: aws:SourceOrgPaths
- The confused deputy problem

Suggestion – Irrelevant condition key in policy

Issue code: IRRELEVANT_CONDITION_KEY_IN_POLICY

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Irrelevant condition key in policy: The condition key {{condition-key}} is not relevant
 for the {{resource-type}} policy.  Use this key in an identity-based policy to govern
 access to this resource.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The condition key {{condition-key}} is not relevant for the
 {{resource-type}} policy.  Use this key in an identity-based policy to govern access
 to this resource."
```

Resolving the suggestion

Some condition keys aren't relevant for resource-based policies. For example, the `s3:ResourceAccount` condition key isn't relevant for the resource-based policy attached to an Amazon S3 bucket or Amazon S3 access point resource type.

You should use the condition key in an identity-based policy to control access to the resource.

Related terms

- Identity-based policies and resource-based policies

Suggestion – Redundant key due to wildcard in condition

Issue code: REDUNDANT_KEY_DUE_TO_WILDCARD_IN_CONDITION

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant key due to wildcard in condition: The key {{key}} in the condition block of
 {{operator}} is redundant because it is always matched. Remove this key to simplify
 the condition.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The key {{key}} in the condition block of {{operator}} is redundant
 because it is always matched. Remove this key to simplify the condition."
```

Resolving the suggestion

Remove the redundant condition key from your policy. The key is always matched due to the wildcard pattern, making it unnecessary. Simplify your condition block by removing this key while maintaining the same effective permissions.

Related terms

- IAM policy elements: Condition
- Conditions with multiple context keys or values
- AWS global condition context keys

Suggestion – Redundant principal in role trust policy

Issue code: REDUNDANT_PRINCIPAL_IN_ROLE_TRUST_POLICY

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant principal in role trust policy: The assumed-role principal
 {{redundant_principal}} is redundant with its parent role {{parent_role}}. Remove the
 assumed-role principal.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The assumed-role principal {{redundant_principal}} is redundant with
 its parent role {{parent_role}}. Remove the assumed-role principal."
```

Resolving the suggestion

If you specify both an assumed-role principal and its parent role in the `Principal` element of a policy, it does not allow or deny any different permissions. For example, it is redundant if you specify the `Principal` element using the following format:

```
"Principal": {
            "AWS": [
            "arn:aws:iam::AWS-account-ID:role/rolename",
            "arn:aws:iam::AWS-account-ID:assumed-role/rolename/rolesessionname"
        ]
```

We recommend removing the assumed-role principal.

Related terms

- Role session principals

Suggestion – Redundant statement due to wildcard in condition

Issue code: REDUNDANT_STATEMENT_DUE_TO_WILDCARD_IN_CONDITION

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Redundant statement due to wildcard in condition: The key {{key}} in the condition
 block of {{operator}} does not match any values. Remove this key to simplify the
 condition.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The key {{key}} in the condition block of {{operator}} does not
 match any values. Remove this key to simplify the condition."
```

Resolving the suggestion

Remove the condition key that doesn't match any values. This key creates an unreachable condition that will never be satisfied, making it redundant. Clean up your policy by removing this key to improve readability and performance.

Related terms

- IAM policy elements: Condition
- Conditions with multiple context keys or values
- AWS global condition context keys

Suggestion – Confirm audience claim type

Issue code: CONFIRM_AUDIENCE_CLAIM_TYPE

Finding type: SUGGESTION

Finding details

In the AWS Management Console, the finding for this check includes the following message:

```
Confirm audience claim type: The "{{key}}" ({{audienceType}}) claim key identifies
 the recipients that the JSON web token is intended for. Because this claim is single-
valued, do not use a qualifier.
```

In programmatic calls to the AWS CLI or AWS API, the finding for this check includes the following message:

```
"findingDetails": "The "{{key}}" ({{audienceType}}) claim key identifies the recipients
 that the JSON web token is intended for. Because this claim is single-valued, do not
 use a qualifier."
```

Resolving the suggestion

The aud (audience) claim key is a unique identifier for your app that is issued to you when you register your app with the IdP and identifies the recipients that the JSON web token is intended for. Audience claims can be multivalued or single-valued. If the claim is multivalued, use a `ForAllValues` or `ForAnyValue` condition set operator. If the claim is single-valued, do not use a condition set operator.

Related terms

- Creating a role for web identity or OpenID Connect Federation (console)
- Multivalued context keys
- Single-valued vs. multivalued condition keys

Validate policies with IAM Access Analyzer custom policy checks

You can use custom policy checks to check for new access based on your security standards. A charge is associated with each check for new access. For more details about pricing, see IAM Access Analyzer pricing.

Validating policies with custom policy checks (console)

As an optional step, you can run a custom policy check when editing a policy in the JSON policy editor in the IAM console. You can check whether the updated policy grants new access compared to the existing version.

To check for new access when editing IAM JSON policies

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.
3. In the list of policies, choose the policy name of the policy that you want to edit. You can use the search box to filter the list of policies.
4. Choose the **Permissions** tab, and then choose **Edit**.

5. Choose the **JSON** option and make updates to your policy.
6. In the policy validation pane below the policy, choose the **Check for new access** tab and then choose **Check policy**. If the modified permissions grant new access, the statement will be highlighted in the policy validation pane.
7. If you don't intend to grant new access, update the policy statement and choose **Check policy** until no new access is detected.

> **Note**
>
> A charge is associated with each check for new access. For more details about pricing, see IAM Access Analyzer pricing.

8. Choose **Next**.
9. On the **Review and save** page, review **Permissions defined in this policy** and then choose **Save changes**.

Validating policies with custom policy checks (AWS CLI or API)

You can run IAM Access Analyzer custom policy checks from the AWS CLI or the IAM Access Analyzer API.

To run IAM Access Analyzer custom policy checks (AWS CLI)

- To check whether new access is allowed for an updated policy when compared to the existing policy, run the following command: check-no-new-access
- To check whether the specified access isn't allowed by a policy, run the following command: check-access-not-granted
- To check whether a resource policy can grant public access to a specified resource type, run the following command: check-no-public-access

To run IAM Access Analyzer custom policy checks (API)

- To check whether new access is allowed for an updated policy when compared to the existing policy, use the CheckNoNewAccess API operation.
- To check whether the specified access isn't allowed by a policy, use the CheckAccessNotGranted API operation.

- To check whether a resource policy can grant public access to a specified resource type, use the CheckNoPublicAccess API operation.

IAM Access Analyzer policy generation

As an administrator or developer, you might grant permissions to IAM entities (users or roles) beyond what they require. IAM provides several options to help you refine the permissions that you grant.

One option is to generate an IAM policy that is based on access activity for an entity. IAM Access Analyzer reviews your AWS CloudTrail logs and generates a policy template that contains the permissions that the entity used in your specified date range. You can use the template to create a policy with fine-grained permissions that grant only the permissions that are required to support your specific use case.

Topics

- How policy generation works
- Service and action-level information
- Things to know about generating policies
- Permissions required to generate a policy
- Generate a policy based on CloudTrail activity (console)
- Generate a policy using AWS CloudTrail data in another account
- Generate a policy based on CloudTrail activity (AWS CLI)
- Generate a policy based on CloudTrail activity (AWS API)
- IAM Access Analyzer policy generation services

How policy generation works

IAM Access Analyzer analyzes your CloudTrail events to identify actions and services that have been used by an IAM entity (user or role). It then generates an IAM policy that is based on that activity. You can refine an entity's permissions when you replace a broad permissions policy attached to the entity with the generated policy. The following is a high-level overview of the policy generation process.

- **Set up for policy template generation** – You specify a time period of up to 90 days for IAM Access Analyzer to analyze your historical AWS CloudTrail events. You must specify an existing service role or create a new one. The service role gives IAM Access Analyzer access to your CloudTrail trail and service last accessed information to identify the services and actions that were used. You must specify the CloudTrail trail that is logging events for the account before you can generate a policy. For more information about IAM Access Analyzer quotas for CloudTrail data, see IAM Access Analyzer quotas.
- **Generate policy** – IAM Access Analyzer generates a policy based on the access activity in your CloudTrail events.
- **Review and customize policy** – After the policy is generated, you can review the services and actions that were used by the entity during the specified date range. You can further customize the policy by adding or removing permissions, specifying resources, and adding conditions to the policy template.
- **Create and attach policy** – You have the option to save the generated policy by creating a managed policy. You can attach the policy that you create to the user or role whose activity was used to generate the policy.

Service and action-level information

When IAM Access Analyzer generates an IAM policy, information is returned to help you to further customize the policy. Two categories of information can be returned when a policy is generated:

- **Policy with action-level information –** For some AWS services, such as Amazon EC2, IAM Access Analyzer can identify the actions found in your CloudTrail events and lists the actions used in the policy it generates. For a list of supported services, see IAM Access Analyzer policy generation services. For some services, IAM Access Analyzer prompts you to add actions for the services to the generated policy.
- **Policy with service-level information –** IAM Access Analyzer uses last accessed information to create a policy template with all of the recently used services. When using the AWS Management Console, we prompt you to review the services and add actions to complete the policy.

For a list of actions in each service, see Actions, Resources, and Condition Keys for AWS Services in the Service Authorization Reference.

Things to know about generating policies

Before you generate a policy, review the following important details.

- **Enable a CloudTrail trail** – You must have a CloudTrail trail enabled for your account to generate a policy based on access activity. When you create a CloudTrail trail, CloudTrail sends events related to your trail to an Amazon S3 bucket that you specify. To learn how to create a CloudTrail trail, see Creating a trail for your AWS account in the *AWS CloudTrail User Guide*.
- **Data events not available** – IAM Access Analyzer does not identify action-level activity for data events, such as Amazon S3 data events, in generated policies.
- **PassRole** – The `iam:PassRole` action is not tracked by CloudTrail and is not included in generated policies.
- **Reduce policy generation time** – To generate a policy faster, reduce the date range that you specify during setup for policy generation.
- **Use CloudTrail for auditing** – Do not use policy generation for auditing purposes; use CloudTrail instead. For more information about using CloudTrail, see Logging IAM and AWS STS API calls with AWS CloudTrail.
- **Denied actions** – Policy generation reviews all CloudTrail events, including denied actions.
- **One policy IAM console** – You can have one generated policy at a time in the IAM console.
- **Generated policy availability IAM console** – You can review a generated policy in the IAM console for up to 7 days after it is generated. After 7 days, you must generate a new policy.
- **Policy generation quotas** – For additional information about IAM Access Analyzer policy generation quotas, see IAM Access Analyzer quotas.
- **Amazon S3 standard rates apply** – When you use the policy generation feature, IAM Access Analyzer reviews CloudTrail logs in your S3 bucket. There are no additional storage charges to access your CloudTrail logs for policy generation. AWS charges standard Amazon S3 rates for requests and data transfer of CloudTrail logs stored in your S3 bucket.
- **AWS Control Tower support** – Policy generation does not support using AWS CloudTrail trails that AWS Control Tower creates when generating policies, for these reasons:
  - The CloudTrail data from the organization is logged in another account, the AWS Control Tower Log Archive account.
  - Permissions for the S3 bucket where these logs are stored cannot be reconfigured due to the restrictions on the S3 logging bucket set by AWS Control Tower's service control policies (SCPs).

Permissions required to generate a policy

The permissions that you need to generate a policy for the first time differ from those that you need to generate a policy for subsequent uses. For more information, see Getting started with AWS Identity and Access Management Access Analyzer.

First-time setup

When you generate a policy for the first time, you must choose a suitable existing service role in your account or create a new service role. The service role gives IAM Access Analyzer access to CloudTrail and service last accessed information in your account. Only administrators should have the permissions necessary to create and configure roles. Therefore, we recommend that an administrator creates the service role during the first-time setup. To learn more about the permissions required to create service roles, see Creating a role to delegate permissions to an AWS service.

Permissions required for service role

When you create a service role, you configure two policies for the role. You attach an IAM *permissions policy* to the role that specifies what the role can do. You also attach a *role trust policy* to the role that specifies the principal who can use the role.

The first example policy shows the permissions policy for the service role that is required to generate a policy. The second example policy shows the role trust policy that is required for the service role. You can use these policies to help you create a service role when you use the AWS API or AWS CLI to generate a policy. When you use the IAM console to create a service role as part of the policy generation process, we generate these policies for you.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cloudtrail:GetTrail",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iam:GetServiceLastAccessedDetails",
                "iam:GenerateServiceLastAccessedDetails"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

The following example policy shows the role trust policy with the permissions that allows IAM Access Analyzer to assume the role.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "access-analyzer.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Subsequent uses

To generate policies in the AWS Management Console, an IAM user must have a permissions policy that allows them to pass the service role that is used for policy generation to IAM Access Analyzer. `iam:PassRole` is usually accompanied by `iam:GetRole` so that the user can get the details of the role to be passed. In this example, the user can pass only roles that exist in the specified account with names that begin with `AccessAnalyzerMonitorServiceRole*`. To learn more about passing IAM roles to AWS services, see Granting a user permissions to pass a role to an AWS service.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUserToPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/service-role/
AccessAnalyzerMonitorServiceRole*"
    }
  ]
}
```

You must also have the following IAM Access Analyzer permissions to generate policies in the AWS Management Console, AWS API, or AWS CLI as shown in the following policy statement.

```
{
  "Sid": "AllowUserToGeneratePolicy",
  "Effect": "Allow",
  "Action": [
    "access-analyzer:CancelPolicyGeneration",
    "access-analyzer:GetGeneratedPolicy",
    "access-analyzer:ListPolicyGenerations",
    "access-analyzer:StartPolicyGeneration"
  ],
  "Resource": "*"
}
```

For both first-time and subsequent uses

When you use the AWS Management Console to generate a policy, you must have `cloudtrail:ListTrails` permission to list the CloudTrail trails in your account as shown in the following policy statement.

```
{
  "Sid": "AllowUserToListTrails",
  "Effect": "Allow",
  "Action": [
    "cloudtrail:ListTrails"
  ],
  "Resource": "*"
}
```

Generate a policy based on CloudTrail activity (console)

You can generate a policy for an IAM user or role.

Step 1: Generate a policy based on CloudTrail activity

The following procedure explains how to generate a policy for a role using the AWS Management Console.

Generate a policy for an IAM role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Roles**.

   > **Note**
   >
   > The steps to generate a policy based on activity for an IAM user are almost identical. To do this, choose **Users** instead of **Roles**.

3. In the list of roles in your account, choose the name of the role whose activity you want to use to generate a policy.
4. On the **Permissions** tab, in the **Generate policy based on CloudTrail events** section, choose **Generate policy**.

5. On the **Generate policy** page, specify the time period that you want IAM Access Analyzer to analyze your CloudTrail events for actions taken with the role. You can choose a range of up to 90 days. We recommend that you choose the shortest time period possible to reduce the policy generation time.
6. In the **CloudTrail access** section, choose a suitable existing role or create a new role if a suitable role does not exist. The role gives IAM Access Analyzer permissions to access your CloudTrail data on your behalf to review access activity to identify the services and actions that have been used. To learn more about the permissions required for this role, see Permissions required to generate a policy.
7. In the **CloudTrail trail to be analyzed** section, specify the CloudTrail trail that logs events for the account.

   If you choose a CloudTrail trail that stores logs in a different account, an information box about cross-account access is displayed. Cross-account access requires additional set up. To learn more, see Choose a role for cross-account access later in this topic.
8. Choose **Generate policy**.
9. While policy generation is in progress, you are returned to the **Roles Summary** page on the **Permissions** tab. Wait until the status in the **Policy request details** section displays **Success**, and then choose **View generated policy**. You can view the generated policy for up to seven days. If you generate another policy, the existing policy is replaced with the new one that you generate.

Step 2: Review permissions and add actions for services used

Review the services and actions that IAM Access Analyzer identified that the role used. You can add actions for any services that were used to the generated policy template.

1. Review the following sections:

   - On the **Review permissions** page, review the list of **Actions included in the generated policy**. The list displays the services *and* actions that IAM Access Analyzer identified were used by the role in the specified date range.
   - The **Services used** section displays additional services that IAM Access Analyzer identified that were used by the role in the specified date range. Information about which actions were used might not be available for the services listed in this section. Use the menus for each service listed to manually choose the actions that you want to include in the policy.

2. When you are done adding actions, choose **Next**.

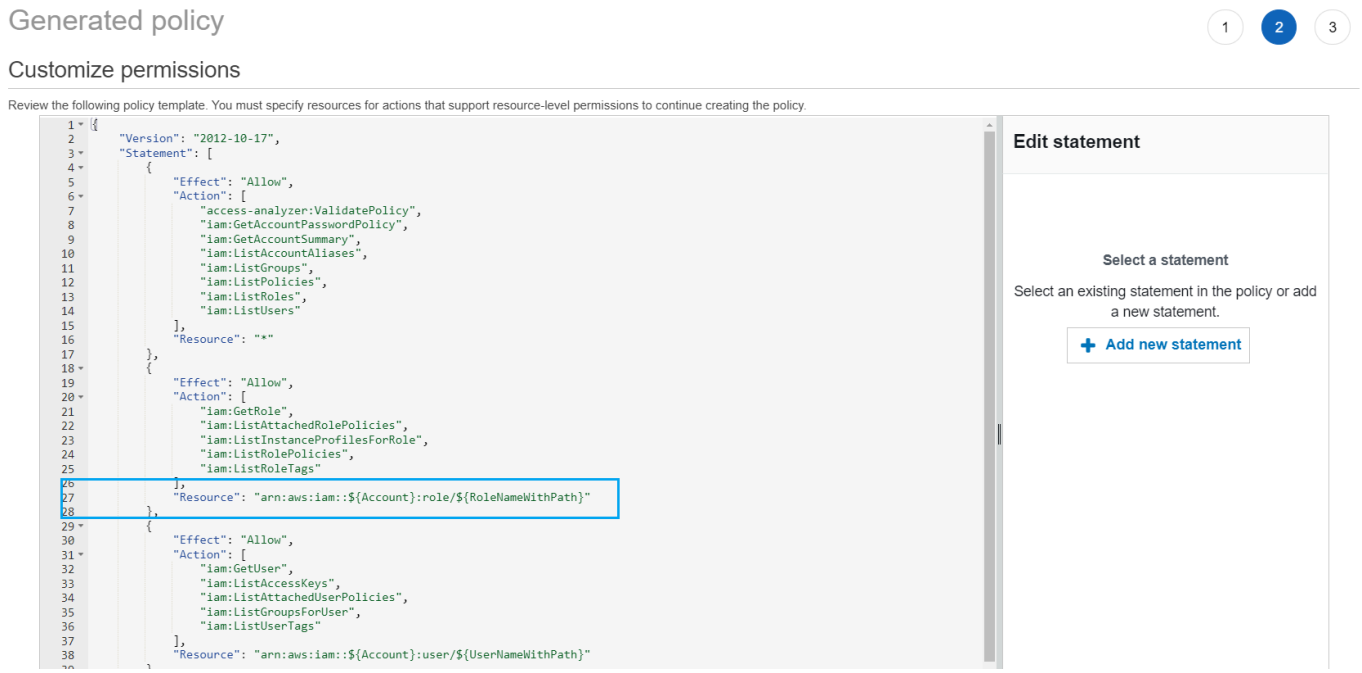
Step 3: Further customize the generated policy

You can further customize the policy by adding or removing permissions or specifying resources.

To customize the generated policy

1. Update the policy template. The policy template contains resource ARN placeholders for actions that support resource-level permissions, as shown in the following image. *Resource-level permissions* refers to the ability to specify which resources users are allowed to perform actions on. We recommend that you use ARNs to specify your individual resources in the policy for actions that support resource-level permissions. You can replace the placeholder resource ARNs with valid resource ARNs for your use case.

   If an action does not support resource-level permissions, you must use a wildcard (*) to specify that all resources can be affected by the action. To learn which AWS services support resource-level permissions, see AWS services that work with IAM. For a list of actions in each service, and to learn which actions support resource-level permissions, see Actions, Resources, and Condition Keys for AWS Services.



2. (Optional) Add, modify, or remove JSON policy statements in the template. To learn more about writing JSON policies, see Creating IAM policies (console).
3. When you are done customizing the policy template, you have the following options:

- (Optional) You can copy the JSON in the template to use separately outside of the **Generated policy** page. For example, if you want to use the JSON to create a policy in a different account. If the policy in your template exceeds the 6,144 character limit for JSON policies, the policy is split into multiple policies.
- Choose **Next** to review and create a managed policy in the same account.

Step 4: Review and create a managed policy

If you have permissions to create and attach IAM policies, you can create a managed policy from the policy that was generated. You can then attach the policy to a user or role in your account.

To review and create a policy

1. On the **Review and create managed policy** page, enter a **Name** and **Description** (optional) for the policy that you are creating.
2. (Optional) In the **Summary** section, you can review the permissions that will be included in the policy.
3. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM resources.
4. When you are finished, do one of the following:
   - You can attach the new policy directly to the role that was used to generate the policy. To do this, near the bottom of the page, select the checkbox next to the **Attach policy to *`YourRoleName`***. Then choose **Create and attach policy**.
   - Otherwise, choose **Create policy**. You can find the policy that you created in the list of policies in the **Policies** navigation pane of the IAM console.
5. You can attach the policy that you created to an entity in your account. After you attach the policy, you can remove any other overly broad policies that might be attached to the entity. To learn how to attach a managed policy, see Adding IAM identity permissions (console).

Generate a policy using AWS CloudTrail data in another account

You might create CloudTrail trails that store data in central accounts to streamline governing activities. For example, you can use AWS Organizations to create a trail that logs all events for all of the AWS accounts in that organization. The trail belongs to a central account.

If you want to generate a policy for a user or role in an account that is different from the account where your CloudTrail log data is stored, you must grant cross-account access. To do this, you need both a role and a bucket policy that grant IAM Access Analyzer permissions to your CloudTrail logs. For more information about creating Organizations trails, see Creating a trail for an organization.

In this example, assume that you want to generate a policy for a user or role in account A. The CloudTrail trail in account A stores CloudTrail logs in a bucket in account B. Before you can generate a policy, you must make the following updates:

1. Choose an existing role, or create a new service role that grants IAM Access Analyzer access to the bucket in account B (where your CloudTrail logs are stored).
2. Verify your Amazon S3 bucket object ownership and bucket permissions policy in account B so that IAM Access Analyzer can access objects in the bucket.

Step 1: Choose or create a role for cross-account access

- On the **Generate policy** screen, the option to **Use an existing role** is pre-selected for you if a role with the required permissions exists in your account. Otherwise, choose **Create and use a new service role**. The new role is used to grant IAM Access Analyzer access to your CloudTrail logs in account B.

Step 2: Verify or update your Amazon S3 bucket configuration in account B

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the **Buckets** list, choose the name of the bucket where your CloudTrail trail logs are stored.
3. Choose the **Permissions** tab and go to the **Object Ownership** section.

   Use Amazon S3 Object Ownership bucket settings to control ownership of objects that you upload to your buckets. By default, when other AWS accounts upload objects to your bucket, the uploading account owns the objects. To generate a policy, the bucket owner must own all of the objects in the bucket. Depending on your ACL use case, you might need to change the **Object Ownership** setting for your bucket. Set **Object Ownership** to one of the following options.

   - **Bucket owner enforced** (recommended)
   - **Bucket owner preferred**

> **Important**
>
> To successfully generate a policy, the objects in the bucket must be owned by the bucket owner. If you choose to use **Bucket owner preferred**, you can only generate a policy for the time period after the object ownership change was made.

To learn more about object ownership in Amazon S3, see Controlling ownership of objects and disabling ACLs for your bucket in the *Amazon S3 User Guide*.

4. Add permissions to your Amazon S3 bucket policy in account B to allow access for the role in account A.

   The following example policy allows `ListBucket` and `GetObject` for the bucket named `amzn-s3-demo-bucket`. It allows access if the role accessing the bucket belongs to an account in your organization and has a name that starts with `AccessAnalyzerMonitorServiceRole`. Using `aws:PrincipalArn` as a `Condition` in the `Resource` element ensures that the role can only access activity for the account if it belongs to account A. You can replace `amzn-s3-demo-bucket` with your bucket name, `optional-prefix` with an optional prefix for the bucket, and `organization-id` with your organization ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PolicyGenerationBucketPolicy",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/optional-prefix/AWSLogs/organization-id/
${aws:PrincipalAccount}/*"
```

```
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "organization-id"
        },
        "ArnLike": {
          "aws:PrincipalArn": "arn:aws:iam::${aws:PrincipalAccount}:role/service-
role/AccessAnalyzerMonitorServiceRole*"
        }
      }
    }
  ]
}
```

5. If you encrypt your logs using AWS KMS, update your AWS KMS key policy in the account where you store the CloudTrail logs to grant IAM Access Analyzer access to use your key, as shown in the following policy example. Replace `CROSS_ACCOUNT_ORG_TRAIL_FULL_ARN` with the ARN for your trail and `organization-id` with your organization ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "kms:Decrypt",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:cloudtrail:arn":
 "CROSS_ACCOUNT_ORG_TRAIL_FULL_ARN",
          "aws:PrincipalOrgID": "organization-id"
        },
        "StringLike": {
          "kms:ViaService": [
            "access-analyzer.*.amazonaws.com",
            "s3.*.amazonaws.com"
            ]
        },
        "ArnLike": {
```

```
          "aws:PrincipalArn": "arn:aws:iam::${aws:PrincipalAccount}:role/service-
role/AccessAnalyzerMonitorServiceRole*"
        }
      }
     }
    ]
}
```

Generate a policy based on CloudTrail activity (AWS CLI)

You can use the following commands to generate a policy using the AWS CLI.

To generate a policy

- aws accessanalyzer start-policy-generation

To view a generated policy

- aws accessanalyzer get-generated-policy

To cancel a policy generation request

- aws accessanalyzer cancel-policy-generation

To view a list of policy generation requests

- aws accessanalyzer list-policy-generations

Generate a policy based on CloudTrail activity (AWS API)

You can use the following operations to generate a policy using the AWS API.

To generate a policy

- StartPolicyGeneration

To view a generated policy

- GetGeneratedPolicy

To cancel a policy generation request

- CancelPolicyGeneration

To view a list of policy generation requests

- ListPolicyGenerations

IAM Access Analyzer policy generation services

The following table lists the AWS services for which IAM Access Analyzer generates policies with action-level information. For a list of actions in each service, see Actions, resources, and condition keys for AWS services in the Service Authorization Reference.

AWS provides policy generation information in JSON format to streamline the automation of policy management workflows. With the service reference information, you can access policy generation information across AWS services from machine-readable files. For more information, see Simplified AWS service information for programmatic access in the Service Authorization Reference.

Service	Service prefix
AWS Identity and Access Management Access Analyzer	access-analyzer
AWS Account Management	account
AWS Certificate Manager	acm
Amazon Managed Workflows for Apache Airflow	airflow
AWS Amplify	amplify
AWS Amplify UI Builder	amplifyuibuilder
Amazon AppIntegrations	app-integrations

Service	Service prefix
AWS AppConfig	appconfig
Amazon AppFlow	appflow
AWS Application Cost Profiler	application-cost-profiler
Amazon CloudWatch Application Insights	applicationinsights
AWS App Mesh	appmesh
Amazon WorkSpaces Applications	appstream
AWS AppSync	appsync
Amazon Managed Service for Prometheus	aps
Amazon Athena	athena
AWS Audit Manager	auditmanager
AWS Auto Scaling	autoscaling
AWS Marketplace	aws-marketplace
AWS Backup	backup
AWS Batch	batch
Amazon Braket	braket
AWS Budgets	budgets
AWS Cloud9	cloud9
AWS CloudFormation	cloudformation
Amazon CloudFront	cloudfront
AWS CloudHSM	cloudhsm

Service	Service prefix
Amazon CloudSearch	cloudsearch
AWS CloudTrail	cloudtrail
Amazon CloudWatch	cloudwatch
AWS CodeArtifact	codeartifact
AWS CodeDeploy	codedeploy
Amazon CodeGuru Profiler	codeguru-profiler
Amazon CodeGuru Reviewer	codeguru-reviewer
AWS CodePipeline	codepipeline
AWS CodeStar	codestar
AWS CodeStar Notifications	codestar-notifications
Amazon Cognito Identity	cognito-identity
Amazon Cognito user pools	cognito-idp
Amazon Cognito Sync	cognito-sync
Amazon Comprehend Medical	comprehendmedical
AWS Compute Optimizer	compute-optimizer
AWS Config	config
Amazon Connect Customer	connect
AWS Cost and Usage Report	cur
AWS Glue DataBrew	databrew

Service	Service prefix
AWS Data Exchange	dataexchange
AWS Data Pipeline	datapipeline
DynamoDB Accelerator	dax
AWS Device Farm	devicefarm
Amazon DevOps Guru	devops-guru
AWS Direct Connect	directconnect
Amazon Data Lifecycle Manager	dlm
AWS Database Migration Service	dms
Amazon DocumentDB Elastic Clusters	docdb-elastic
Amazon DynamoDB	dynamodb
Amazon Elastic Block Store	ebs
Amazon Elastic Compute Cloud	ec2
Amazon Elastic Container Registry	ecr
Amazon Elastic Container Registry Public	ecr-public
Amazon Elastic Container Service	ecs
Amazon Elastic Kubernetes Service	eks
Amazon ElastiCache	elasticache
AWS Elastic Beanstalk	elasticbeanstalk
Amazon Elastic File System	elasticfilesystem
Elastic Load Balancing	elasticloadbalancing

Service	Service prefix
Amazon EMR on EKS (EMR Containers)	emr-containers
Amazon EMR Serverless	emr-serverless
Amazon OpenSearch Service	es
Amazon EventBridge	events
Amazon FinSpace	finspace
Amazon Data Firehose	firehose
AWS Fault Injection Service	fis
AWS Firewall Manager	fms
Amazon Fraud Detector	frauddetector
Amazon FSx	fsx
Amazon GameLift Servers	gamelift
Amazon Location Service	geo
Amazon Glacier	glacier
Amazon Managed Grafana	grafana
AWS IoT Greengrass	greengrass
AWS Ground Station	groundstation
Amazon GuardDuty	guardduty
AWS HealthLake	healthlake
Amazon Honeycode	honeycode
AWS Identity and Access Management	iam

Service	Service prefix
AWS Identity Store	identitystore
EC2 Image Builder	imagebuilder
Amazon Inspector Classic	inspector
Amazon Inspector	inspector2
AWS IoT	iot
AWS IoT Analytics	iotanalytics
AWS IoT Core Device Advisor	iotdeviceadvisor
AWS IoT Events	iotevents
AWS IoT Fleet Hub	iotfleethub
AWS IoT SiteWise	iotsitewise
AWS IoT TwinMaker	iottwinmaker
AWS IoT Wireless	iotwireless
Amazon Interactive Video Service	ivs
Amazon Interactive Video Service Chat	ivschat
Amazon Managed Streaming for Apache Kafka	kafka
Amazon Managed Streaming for Kafka Connect	kafkaconnect
Amazon Kendra	kendra
Amazon Kinesis	kinesis
Amazon Kinesis Analytics V2	kinesisanalytics
AWS Key Management Service	kms

Service	Service prefix
AWS Lambda	lambda
Amazon Lex V2	lex
AWS License Manager Linux Subscriptions Manager	license-manager-linux-subscriptions
Amazon Lightsail	lightsail
Amazon CloudWatch Logs	logs
Amazon Lookout for Equipment	lookoutequipment
Amazon Lookout for Metrics	lookoutmetrics
AWS Mainframe Modernization	m2
Amazon Managed Blockchain	managedblockchain
AWS Elemental MediaConnect	mediaconnect
AWS Elemental MediaConvert	mediaconvert
AWS Elemental MediaLive	medialive
AWS Elemental MediaTailor	mediatailor
Amazon MemoryDB	memorydb
AWS Transform MGN	mgn
AWS Migration Hub	mgh
AWS Migration Hub Strategy Recommendations	migrationhub-strategy
Amazon Pinpoint	mobiletargeting
Amazon MQ	mq

Service	Service prefix
AWS Network Manager	networkmanager
Amazon Nimble Studio	nimble
AWS HealthOmics	omics
AWS OpsWorks	opsworks
AWS OpsWorks CM	opsworks-cm
AWS Outposts	outposts
AWS Organizations	organizations
AWS Performance Insights	pi
Amazon EventBridge Pipes	pipes
Amazon Polly	polly
Amazon Connect Customer Customer Profiles	profile
Amazon QLDB	qldb
AWS Resource Access Manager	ram
AWS Recycle Bin	rbin
Amazon Relational Database Service	rds
Amazon Redshift	redshift
Amazon Redshift Data API	redshift-data
AWS Migration Hub Refactor Spaces	refactor-spaces
Amazon Rekognition	rekognition
AWS Resilience Hub	resiliencehub

Service	Service prefix
AWS Resource Explorer	resource-explorer-2
AWS Resource Groups	resource-groups
AWS RoboMaker	robomaker
AWS Identity and Access Management Roles Anywhere	rolesanywhere
Amazon Route 53	route53
Amazon Route 53 Recovery Controls	route53-recovery-control-config
Amazon Route 53 Recovery Readiness	route53-recovery-readiness
Amazon Route 53 Resolver	route53resolver
AWS CloudWatch RUM	rum
Amazon Simple Storage Service	s3
Amazon S3 on Outposts	s3-outposts
Amazon SageMaker AI geospatial capabilities	sagemaker-geospatial
Savings Plans	savingsplans
Amazon EventBridge Schemas	schemas
Amazon SimpleDB	sdb
AWS Secrets Manager	secretsmanager
AWS Security Hub CSPM	securityhub
Amazon Security Lake	securitylake

Service	Service prefix
AWS Serverless Application Repository	serverlessrepo
AWS Service Catalog	servicecatalog
AWS Cloud Map	servicediscovery
Service Quotas	servicequotas
Amazon Simple Email Service	ses
AWS Shield	shield
AWS Signer	signer
AWS SimSpace Weaver	simspaceweaver
AWS Server Migration Service	sms
Amazon Pinpoint SMS and Voice Service	sms-voice
AWS Snowball Edge	snowball
Amazon Simple Queue Service	sqs
AWS Systems Manager	ssm
AWS Systems Manager Incident Manager	ssm-incidents
AWS Systems Manager for SAP	ssm-sap
AWS Step Functions	states
AWS Security Token Service	sts
Amazon Simple Workflow Service	swf
Amazon CloudWatch Synthetics	synthetics
AWS Resource Groups Tagging API	tag

Service	Service prefix
Amazon Textract	textract
Amazon Timestream	timestream
AWS Telco Network Builder	tnb
Amazon Transcribe	transcribe
AWS Transfer Family	transfer
Amazon Translate	translate
Connect Customer Voice ID	voiceid
Amazon VPC Lattice	vpc-lattice
AWS WAFV2	wafv2
AWS Well-Architected Tool	wellarchitected
Amazon Connect Customer Wisdom	wisdom
Amazon WorkLink	worklink
Amazon WorkSpaces	workspaces
AWS X-Ray	xray

Actions for policy generation information

The following table lists the actions for which IAM Access Analyzer generates policies with action-level information.

Important

The `iam:UpdateAccountName` action will be deprecated on April 22, 2026. After April 22, 2026, only the `account:PutAccountName` permission will control account name update

access. We strongly recommend that you update any service control policies (SCPs) that control account name updates to use the `account:PutAccountName` permission.

Service prefix	Actions
access-analyzer	access-analyzer:ApplyArchiveRule
	access-analyzer:CancelPolicyGeneration
	access-analyzer:CheckAccessNotGranted
	access-analyzer:CheckNoNewAccess
	access-analyzer:CheckNoPublicAccess
	access-analyzer:CreateAccessPreview
	access-analyzer:CreateAnalyzer
	access-analyzer:CreateArchiveRule
	access-analyzer:DeleteAnalyzer
	access-analyzer:DeleteArchiveRule
	access-analyzer:GenerateFindingRecommendation
	access-analyzer:GetAccessPreview
	access-analyzer:GetAnalyzedResource
	access-analyzer:GetAnalyzer
	access-analyzer:GetArchiveRule
	access-analyzer:GetFinding
	access-analyzer:GetFindingRecommendation
	access-analyzer:GetFindingsStatistics

Service prefix	Actions
	access-analyzer:GetGeneratedPolicy
	access-analyzer:ListAccessPreviewFindings
	access-analyzer:ListAccessPreviews
	access-analyzer:ListAnalyzedResources
	access-analyzer:ListAnalyzers
	access-analyzer:ListArchiveRules
	access-analyzer:ListFindings
	access-analyzer:ListPolicyGenerations
	access-analyzer:StartPolicyGeneration
	access-analyzer:StartResourceScan
	access-analyzer:UpdateAnalyzer
	access-analyzer:UpdateArchiveRule
	access-analyzer:UpdateFindings
	access-analyzer:ValidatePolicy

Service prefix	Actions
account	account:AcceptPrimaryEmailUpdate account:DeleteAlternateContact account:DisableRegion account:EnableRegion account:GetAccountInformation account:GetAlternateContact account:GetContactInformation account:GetGovCloudAccountInformation account:GetPrimaryEmail account:GetRegionOptStatus account:ListRegions account:PutAccountName account:PutAlternateContact account:PutContactInformation account:StartPrimaryEmailUpdate

Service prefix	Actions
acm	acm:CreateAcmeEndpoint
	acm:DeleteCertificate
	acm:DescribeCertificate
	acm:ExportCertificate
	acm:GetAccountConfiguration
	acm:GetCertificate
	acm:ImportCertificate
	acm:ListAcmeEndpoints
	acm:ListCertificates
	acm:PutAccountConfiguration
	acm:RenewCertificate
	acm:RequestCertificate
	acm:ResendValidationEmail
	acm:SearchCertificates
	acm:UpdateCertificateOptions

Service prefix	Actions
airflow	airflow:CreateCliToken airflow:CreateEnvironment airflow:CreateWebLoginToken airflow:DeleteEnvironment airflow:GetEnvironment airflow:ListEnvironments airflow:PublishMetrics airflow:UpdateEnvironment

Service prefix	Actions
amplify	amplify:CreateApp
	amplify:CreateBackendEnvironment
	amplify:CreateBranch
	amplify:CreateDeployment
	amplify:CreateDomainAssociation
	amplify:CreateWebHook
	amplify:DeleteApp
	amplify:DeleteBackendEnvironment
	amplify:DeleteBranch
	amplify:DeleteDomainAssociation
	amplify:DeleteJob
	amplify:DeleteWebHook
	amplify:GenerateAccessLogs
	amplify:GetApp
	amplify:GetArtifactUrl
	amplify:GetBackendEnvironment
	amplify:GetBranch
	amplify:GetDomainAssociation
	amplify:GetJob
	amplify:GetWebHook
	amplify:ListApps

Service prefix	Actions
	amplify:ListArtifacts amplify:ListBackendEnvironments amplify:ListBranches amplify:ListDomainAssociations amplify:ListJobs amplify:ListWebHooks amplify:StartDeployment amplify:StartJob amplify:StopJob amplify:UpdateApp amplify:UpdateBranch amplify:UpdateDomainAssociation amplify:UpdateWebHook

Service prefix	Actions
amplifyuibuilder	amplifyuibuilder:CreateComponent amplifyuibuilder:CreateForm amplifyuibuilder:CreateTheme amplifyuibuilder:DeleteComponent amplifyuibuilder:DeleteForm amplifyuibuilder:DeleteTheme amplifyuibuilder:ExportComponents amplifyuibuilder:ExportThemes amplifyuibuilder:GetCodegenJob amplifyuibuilder:ListCodegenJobs amplifyuibuilder:ListComponents amplifyuibuilder:ListForms amplifyuibuilder:ListThemes amplifyuibuilder:ResetMetadataFlag amplifyuibuilder:StartCodegenJob amplifyuibuilder:UpdateComponent amplifyuibuilder:UpdateForm amplifyuibuilder:UpdateTheme

Service prefix	Actions
app-integrations	app-integrations:CreateApplication
	app-integrations:CreateDataIntegration
	app-integrations:CreateDataIntegrationAssociation
	app-integrations:CreateEventIntegration
	app-integrations:DeleteApplication
	app-integrations:DeleteDataIntegration
	app-integrations:DeleteEventIntegration
	app-integrations:GetApplication
	app-integrations:GetDataIntegration
	app-integrations:GetEventIntegration
	app-integrations:ListApplicationAssociations
	app-integrations:ListApplications
	app-integrations:ListDataIntegrationAssociations
	app-integrations:ListDataIntegrations
	app-integrations:ListEventIntegrationAssociations
	app-integrations:ListEventIntegrations
	app-integrations:UpdateApplication
	app-integrations:UpdateDataIntegration
	app-integrations:UpdateDataIntegrationAssociation
	app-integrations:UpdateEventIntegration

Service prefix	Actions
appconfig	appconfig:CreateApplication appconfig:CreateConfigurationProfile appconfig:CreateDeploymentStrategy appconfig:CreateEnvironment appconfig:CreateExtension appconfig:CreateExtensionAssociation appconfig:CreateHostedConfigurationVersion appconfig:DeleteApplication appconfig:DeleteConfigurationProfile appconfig:DeleteDeploymentStrategy appconfig:DeleteEnvironment appconfig:DeleteExtension appconfig:DeleteExtensionAssociation appconfig:DeleteHostedConfigurationVersion appconfig:GetAccountSettings appconfig:GetApplication appconfig:GetConfiguration appconfig:GetConfigurationProfile appconfig:GetDeployment appconfig:GetDeploymentStrategy appconfig:GetEnvironment

Service prefix	Actions
	appconfig:GetExtension appconfig:GetExtensionAssociation appconfig:GetHostedConfigurationVersion appconfig:ListApplications appconfig:ListConfigurationProfiles appconfig:ListDeploymentStrategies appconfig:ListDeployments appconfig:ListEnvironments appconfig:ListExtensionAssociations appconfig:ListExtensions appconfig:ListHostedConfigurationVersions appconfig:StartDeployment appconfig:StopDeployment appconfig:UpdateAccountSettings appconfig:UpdateApplication appconfig:UpdateConfigurationProfile appconfig:UpdateDeploymentStrategy appconfig:UpdateEnvironment appconfig:UpdateExtension appconfig:UpdateExtensionAssociation appconfig:ValidateConfiguration

Service prefix	Actions
appflow	appflow:CancelFlowExecutions
	appflow:CreateConnectorProfile
	appflow:CreateFlow
	appflow:DeleteConnectorProfile
	appflow:DeleteFlow
	appflow:DescribeConnector
	appflow:DescribeConnectorEntity
	appflow:DescribeConnectorProfiles
	appflow:DescribeConnectors
	appflow:DescribeFlow
	appflow:DescribeFlowExecutionRecords
	appflow:ListConnectorEntities
	appflow:ListConnectors
	appflow:ListFlows
	appflow:RegisterConnector
	appflow:ResetConnectorMetadataCache
	appflow:StartFlow
	appflow:StopFlow
	appflow:UnRegisterConnector
	appflow:UpdateConnectorProfile
	appflow:UpdateConnectorRegistration

Service prefix	Actions
	appflow:UpdateFlow

Service prefix	Actions
applicationinsights	applicationinsights:AddWorkload applicationinsights:CreateApplication applicationinsights:CreateComponent applicationinsights:CreateLogPattern applicationinsights:DeleteApplication applicationinsights:DeleteComponent applicationinsights:DeleteLogPattern applicationinsights:DescribeApplication applicationinsights:DescribeComponent applicationinsights:DescribeComponentConfiguration applicationinsights:DescribeComponentConfigurationRecommendation applicationinsights:DescribeLogPattern applicationinsights:DescribeObservation applicationinsights:DescribeProblem applicationinsights:DescribeProblemObservations applicationinsights:DescribeWorkload applicationinsights:ListApplications applicationinsights:ListComponents applicationinsights:ListConfigurationHistory applicationinsights:ListLogPatternSets

Service prefix	Actions
	applicationinsights:ListLogPatterns applicationinsights:ListProblems applicationinsights:ListWorkloads applicationinsights:RemoveWorkload applicationinsights:UpdateApplication applicationinsights:UpdateComponent applicationinsights:UpdateComponentConfiguration applicationinsights:UpdateLogPattern applicationinsights:UpdateWorkload

Service prefix	Actions
appmesh	appmesh:CreateGatewayRoute
	appmesh:CreateMesh
	appmesh:CreateRoute
	appmesh:CreateVirtualGateway
	appmesh:CreateVirtualNode
	appmesh:CreateVirtualRouter
	appmesh:CreateVirtualService
	appmesh:DeleteGatewayRoute
	appmesh:DeleteMesh
	appmesh:DeleteRoute
	appmesh:DeleteVirtualGateway
	appmesh:DeleteVirtualNode
	appmesh:DeleteVirtualRouter
	appmesh:DeleteVirtualService
	appmesh:DescribeGatewayRoute
	appmesh:DescribeMesh
	appmesh:DescribeRoute
	appmesh:DescribeVirtualGateway
	appmesh:DescribeVirtualNode
	appmesh:DescribeVirtualRouter
	appmesh:DescribeVirtualService

Service prefix	Actions
	appmesh:ListGatewayRoutes
	appmesh:ListMeshes
	appmesh:ListRoutes
	appmesh:ListVirtualGateways
	appmesh:ListVirtualNodes
	appmesh:ListVirtualRouters
	appmesh:ListVirtualServices
	appmesh:StreamAggregatedResources
	appmesh:UpdateGatewayRoute
	appmesh:UpdateMesh
	appmesh:UpdateRoute
	appmesh:UpdateVirtualGateway
	appmesh:UpdateVirtualNode
	appmesh:UpdateVirtualRouter
	appmesh:UpdateVirtualService

Service prefix	Actions
appstream	appstream:AssociateAppBlockBuilderAppBlock appstream:AssociateApplicationFleet appstream:AssociateApplicationToEntitlement appstream:AssociateFleet appstream:AssociateSoftwareToImageBuilder appstream:BatchAssociateUserStack appstream:BatchDisassociateUserStack appstream:CopyImage appstream:CreateAppBlock appstream:CreateAppBlockBuilder appstream:CreateAppBlockBuilderStreamingURL appstream:CreateApplication appstream:CreateDirectoryConfig appstream:CreateEntitlement appstream:CreateFleet appstream:CreateImageBuilder appstream:CreateImageBuilderStreamingURL appstream:CreateStack appstream:CreateStreamingURL appstream:CreateThemeForStack appstream:CreateUpdatedImage

Service prefix	Actions
	appstream:CreateUsageReportSubscription appstream:CreateUser appstream:DeleteAppBlock appstream:DeleteAppBlockBuilder appstream:DeleteApplication appstream:DeleteDirectoryConfig appstream:DeleteEntitlement appstream:DeleteFleet appstream:DeleteImage appstream:DeleteImageBuilder appstream:DeleteImagePermissions appstream:DeleteStack appstream:DeleteThemeForStack appstream:DeleteUsageReportSubscription appstream:DeleteUser appstream:DescribeAppBlockBuilderAppBlockAssociations appstream:DescribeAppBlockBuilders appstream:DescribeAppBlocks appstream:DescribeAppLicenseUsage appstream:DescribeApplicationFleetAssociations appstream:DescribeApplications

Service prefix	Actions
	appstream:DescribeDirectoryConfigs appstream:DescribeEntitlements appstream:DescribeFleets appstream:DescribeImageBuilders appstream:DescribeImagePermissions appstream:DescribeImages appstream:DescribeSessions appstream:DescribeStacks appstream:DescribeThemeForStack appstream:DescribeUsageReportSubscriptions appstream:DescribeUserStackAssociations appstream:DescribeUsers appstream:DisableUser appstream:DisassociateAppBlockBuilderAppBlock appstream:DisassociateApplicationFleet appstream:DisassociateApplicationFromEntitlement appstream:DisassociateFleet appstream:DisassociateSoftwareFromImageBuilder appstream:EnableUser appstream:ExpireSession appstream:GetExportImageTask

Service prefix	Actions
	appstream:ListAssociatedFleets appstream:ListAssociatedStacks appstream:ListEntitledApplications appstream:ListExportImageTasks appstream:StartAppBlockBuilder appstream:StartFleet appstream:StartImageBuilder appstream:StartSoftwareDeploymentToImageBuilder appstream:StopAppBlockBuilder appstream:StopFleet appstream:StopImageBuilder appstream:UpdateAppBlockBuilder appstream:UpdateApplication appstream:UpdateDirectoryConfig appstream:UpdateEntitlement appstream:UpdateFleet appstream:UpdateImagePermissions appstream:UpdateStack appstream:UpdateThemeForStack

Service prefix	Actions
appsync	appsync:AssociateApi
	appsync:AssociateMergedGraphqlApi
	appsync:AssociateSourceGraphqlApi
	appsync:CreateApi
	appsync:CreateApiCache
	appsync:CreateApiKey
	appsync:CreateChannelNamespace
	appsync:CreateDataSource
	appsync:CreateDomainName
	appsync:CreateFunction
	appsync:CreateGraphqlApi
	appsync:CreateResolver
	appsync:CreateType
	appsync:DeleteApi
	appsync:DeleteApiCache
	appsync:DeleteApiKey
	appsync:DeleteChannelNamespace
	appsync:DeleteDataSource
	appsync:DeleteDomainName
	appsync:DeleteFunction
	appsync:DeleteGraphqlApi

Service prefix	Actions
	appsync:DeleteResolver
	appsync:DeleteType
	appsync:DisassociateApi
	appsync:DisassociateMergedGraphqlApi
	appsync:DisassociateSourceGraphqlApi
	appsync:EvaluateCode
	appsync:EvaluateMappingTemplate
	appsync:FlushApiCache
	appsync:GetApi
	appsync:GetApiAssociation
	appsync:GetApiCache
	appsync:GetChannelNamespace
	appsync:GetDataSource
	appsync:GetDataSourceIntrospection
	appsync:GetDomainName
	appsync:GetFunction
	appsync:GetGraphqlApi
	appsync:GetGraphqlApiEnvironmentVariables
	appsync:GetIntrospectionSchema
	appsync:GetResolver
	appsync:GetSchemaCreationStatus

Service prefix	Actions
	appsync:GetSourceApiAssociation
	appsync:GetType
	appsync:ListApiKeys
	appsync:ListApis
	appsync:ListChannelNamespaces
	appsync:ListDataSources
	appsync:ListDomainNames
	appsync:ListFunctions
	appsync:ListGraphqlApis
	appsync:ListResolvers
	appsync:ListResolversByFunction
	appsync:ListSourceApiAssociations
	appsync:ListTypes
	appsync:ListTypesByAssociation
	appsync:PutGraphqlApiEnvironmentVariables
	appsync:StartDataSourceIntrospection
	appsync:StartSchemaCreation
	appsync:StartSchemaMerge
	appsync:UpdateApi
	appsync:UpdateApiCache
	appsync:UpdateApiKey

Service prefix	Actions
	appsync:UpdateChannelNamespace appsync:UpdateDataSource appsync:UpdateDomainName appsync:UpdateFunction appsync:UpdateGraphqlApi appsync:UpdateResolver appsync:UpdateSourceApiAssociation appsync:UpdateType

Service prefix	Actions
aps	aps:CreateAlertManagerDefinition aps:CreateAnomalyDetector aps:CreateLoggingConfiguration aps:CreateQueryLoggingConfiguration aps:CreateRuleGroupsNamespace aps:CreateWorkspace aps:DeleteAlertManagerDefinition aps:DeleteAnomalyDetector aps:DeleteLoggingConfiguration aps:DeleteQueryLoggingConfiguration aps:DeleteResourcePolicy aps:DeleteRuleGroupsNamespace aps:DeleteScraper aps:DeleteScraperLoggingConfiguration aps:DeleteWorkspace aps:DescribeAlertManagerDefinition aps:DescribeAnomalyDetector aps:DescribeLoggingConfiguration aps:DescribeQueryLoggingConfiguration aps:DescribeResourcePolicy aps:DescribeRuleGroupsNamespace

Service prefix	Actions
	aps:DescribeScraper aps:DescribeScraperLoggingConfiguration aps:DescribeWorkspace aps:DescribeWorkspaceConfiguration aps:GetDefaultScraperConfiguration aps:ListAnomalyDetectors aps:ListRuleGroupsNamespaces aps:ListScrapers aps:ListWorkspaces aps:PutAlertManagerDefinition aps:PutAnomalyDetector aps:PutResourcePolicy aps:PutRuleGroupsNamespace aps:UpdateLoggingConfiguration aps:UpdateQueryLoggingConfiguration aps:UpdateScraper aps:UpdateScraperLoggingConfiguration aps:UpdateWorkspaceAlias aps:UpdateWorkspaceConfiguration

Service prefix	Actions
athena	athena:BatchGetNamedQuery
	athena:BatchGetPreparedStatement
	athena:BatchGetQueryExecution
	athena:CancelCapacityReservation
	athena:CreateCapacityReservation
	athena:CreateDataCatalog
	athena:CreateNamedQuery
	athena:CreateNotebook
	athena:CreatePreparedStatement
	athena:CreatePresignedNotebookUrl
	athena:CreateWorkGroup
	athena:DeleteCapacityReservation
	athena:DeleteDataCatalog
	athena:DeleteNamedQuery
	athena:DeleteNotebook
	athena:DeletePreparedStatement
	athena:DeleteWorkGroup
	athena:ExportNotebook
	athena:GetCalculationExecution
	athena:GetCalculationExecutionCode
	athena:GetCalculationExecutionStatus

Service prefix	Actions
	athena:GetCapacityAssignmentConfiguration
	athena:GetCapacityReservation
	athena:GetDataCatalog
	athena:GetDatabase
	athena:GetNamedQuery
	athena:GetNotebookMetadata
	athena:GetPreparedStatement
	athena:GetQueryExecution
	athena:GetQueryResults
	athena:GetQueryResultsStream
	athena:GetQueryRuntimeStatistics
	athena:GetResourceDashboard
	athena:GetSession
	athena:GetSessionEndpoint
	athena:GetSessionStatus
	athena:GetTableMetadata
	athena:GetWorkGroup
	athena:ImportNotebook
	athena:ListApplicationDPUSizes
	athena:ListCalculationExecutions
	athena:ListCapacityReservations

Service prefix	Actions
	athena:ListDataCatalogs
	athena:ListDatabases
	athena:ListEngineVersions
	athena:ListExecutors
	athena:ListNamedQueries
	athena:ListNotebookMetadata
	athena:ListNotebookSessions
	athena:ListPreparedStatements
	athena:ListQueryExecutions
	athena:ListSessions
	athena:ListTableMetadata
	athena:ListWorkGroups
	athena:PutCapacityAssignmentConfiguration
	athena:StartCalculationExecution
	athena:StartQueryExecution
	athena:StartSession
	athena:StopCalculationExecution
	athena:StopQueryExecution
	athena:TerminateSession
	athena:UpdateCapacityReservation
	athena:UpdateDataCatalog

Service prefix	Actions
	athena:UpdateNamedQuery athena:UpdateNotebook athena:UpdateNotebookMetadata athena:UpdatePreparedStatement athena:UpdateWorkGroup

Service prefix	Actions
auditmanager	auditmanager:AssociateAssessmentReportEvidenceFolder auditmanager:BatchAssociateAssessmentReportEvidence auditmanager:BatchCreateDelegationByAssessment auditmanager:BatchDeleteDelegationByAssessment auditmanager:BatchDisassociateAssessmentReportEvidence auditmanager:BatchImportEvidenceToAssessmentControl auditmanager:CreateAssessment auditmanager:CreateAssessmentFramework auditmanager:CreateAssessmentReport auditmanager:CreateControl auditmanager:DeleteAssessment auditmanager:DeleteAssessmentFramework auditmanager:DeleteAssessmentFrameworkShare auditmanager:DeleteAssessmentReport auditmanager:DeleteControl auditmanager:DeregisterAccount auditmanager:DeregisterOrganizationAdminAccount auditmanager:DisassociateAssessmentReportEvidenceFolder auditmanager:GetAccountStatus auditmanager:GetAssessment auditmanager:GetAssessmentFramework

Service prefix	Actions
	auditmanager:GetAssessmentReportUrl auditmanager:GetChangeLogs auditmanager:GetControl auditmanager:GetDelegations auditmanager:GetEvidence auditmanager:GetEvidenceByEvidenceFolder auditmanager:GetEvidenceFileUploadUrl auditmanager:GetEvidenceFolder auditmanager:GetEvidenceFoldersByAssessment auditmanager:GetEvidenceFoldersByAssessmentControl auditmanager:GetInsights auditmanager:GetInsightsByAssessment auditmanager:GetOrganizationAdminAccount auditmanager:GetServicesInScope auditmanager:GetSettings auditmanager:ListAssessmentControlInsightsByControlDomain auditmanager:ListAssessmentFrameworkShareRequests auditmanager:ListAssessmentFrameworks auditmanager:ListAssessmentReports auditmanager:ListAssessments auditmanager:ListControlDomainInsights

Service prefix	Actions
	auditmanager:ListControlDomainInsightsByAssessment auditmanager:ListControlInsightsByControlDomain auditmanager:ListControls auditmanager:ListKeywordsForDataSource auditmanager:ListNotifications auditmanager:RegisterAccount auditmanager:RegisterOrganizationAdminAccount auditmanager:StartAssessmentFrameworkShare auditmanager:UpdateAssessment auditmanager:UpdateAssessmentControl auditmanager:UpdateAssessmentControlSetStatus auditmanager:UpdateAssessmentFramework auditmanager:UpdateAssessmentFrameworkShare auditmanager:UpdateAssessmentStatus auditmanager:UpdateControl auditmanager:UpdateSettings auditmanager:ValidateAssessmentReportIntegrity

Service prefix	Actions
autoscaling	autoscaling:AttachInstances
	autoscaling:AttachLoadBalancerTargetGroups
	autoscaling:AttachLoadBalancers
	autoscaling:AttachTrafficSources
	autoscaling:BatchDeleteScheduledAction
	autoscaling:BatchPutScheduledUpdateGroupAction
	autoscaling:CancelInstanceRefresh
	autoscaling:CompleteLifecycleAction
	autoscaling:CreateAutoScalingGroup
	autoscaling:CreateLaunchConfiguration
	autoscaling:DeleteAutoScalingGroup
	autoscaling:DeleteLaunchConfiguration
	autoscaling:DeleteLifecycleHook
	autoscaling:DeleteNotificationConfiguration
	autoscaling:DeletePolicy
	autoscaling:DeleteScheduledAction
	autoscaling:DeleteWarmPool
	autoscaling:DescribeAccountLimits
	autoscaling:DescribeAdjustmentTypes
	autoscaling:DescribeAutoScalingGroups
	autoscaling:DescribeAutoScalingInstances

Service prefix	Actions
	autoscaling:DescribeAutoScalingNotificationTypes
	autoscaling:DescribeInstanceRefreshes
	autoscaling:DescribeLaunchConfigurations
	autoscaling:DescribeLifecycleHookTypes
	autoscaling:DescribeLifecycleHooks
	autoscaling:DescribeLoadBalancerTargetGroups
	autoscaling:DescribeLoadBalancers
	autoscaling:DescribeMetricCollectionTypes
	autoscaling:DescribeNotificationConfigurations
	autoscaling:DescribePolicies
	autoscaling:DescribeScalingActivities
	autoscaling:DescribeScalingProcessTypes
	autoscaling:DescribeScheduledActions
	autoscaling:DescribeTerminationPolicyTypes
	autoscaling:DescribeTrafficSources
	autoscaling:DescribeWarmPool
	autoscaling:DetachInstances
	autoscaling:DetachLoadBalancerTargetGroups
	autoscaling:DetachLoadBalancers
	autoscaling:DetachTrafficSources
	autoscaling:DisableMetricsCollection

Service prefix	Actions
	autoscaling:EnableMetricsCollection autoscaling:EnterStandby autoscaling:ExecutePolicy autoscaling:ExitStandby autoscaling:GetPredictiveScalingForecast autoscaling:PutLifecycleHook autoscaling:PutNotificationConfiguration autoscaling:PutScalingPolicy autoscaling:PutScheduledUpdateGroupAction autoscaling:PutWarmPool autoscaling:RecordLifecycleActionHeartbeat autoscaling:ResumeProcesses autoscaling:RollbackInstanceRefresh autoscaling:SetDesiredCapacity autoscaling:SetInstanceHealth autoscaling:SetInstanceProtection autoscaling:StartInstanceRefresh autoscaling:SuspendProcesses autoscaling:TerminateInstanceInAutoScalingGroup autoscaling:UpdateAutoScalingGroup
aws-marketplace	aws-marketplace:GetEntitlements

Service prefix	Actions
backup	backup:CancelLegalHold
	backup:CreateBackupPlan
	backup:CreateBackupSelection
	backup:CreateBackupVault
	backup:CreateFramework
	backup:CreateLegalHold
	backup:CreateLogicallyAirGappedBackupVault
	backup:CreateReportPlan
	backup:CreateRestoreAccessBackupVault
	backup:CreateRestoreTestingPlan
	backup:CreateRestoreTestingSelection
	backup:CreateTieringConfiguration
	backup:DeleteBackupPlan
	backup:DeleteBackupSelection
	backup:DeleteBackupVault
	backup:DeleteBackupVaultAccessPolicy
	backup:DeleteBackupVaultLockConfiguration
	backup:DeleteBackupVaultNotifications
	backup:DeleteFramework
	backup:DeleteRecoveryPoint
	backup:DeleteReportPlan

Service prefix	Actions
	backup:DeleteRestoreTestingPlan
	backup:DeleteRestoreTestingSelection
	backup:DeleteTieringConfiguration
	backup:DescribeBackupJob
	backup:DescribeBackupVault
	backup:DescribeCopyJob
	backup:DescribeFramework
	backup:DescribeGlobalSettings
	backup:DescribeProtectedResource
	backup:DescribeRecoveryPoint
	backup:DescribeRegionSettings
	backup:DescribeReportJob
	backup:DescribeReportPlan
	backup:DescribeRestoreJob
	backup:DescribeScanJob
	backup:DisassociateRecoveryPoint
	backup:DisassociateRecoveryPointFromParent
	backup:ExportBackupPlanTemplate
	backup:GetBackupPlan
	backup:GetBackupPlanFromJSON
	backup:GetBackupPlanFromTemplate

Service prefix	Actions
	backup:GetBackupSelection
	backup:GetBackupVaultAccessPolicy
	backup:GetBackupVaultNotifications
	backup:GetLegalHold
	backup:GetRecoveryPointRestoreMetadata
	backup:GetRestoreJobMetadata
	backup:GetRestoreTestingInferredMetadata
	backup:GetRestoreTestingPlan
	backup:GetRestoreTestingSelection
	backup:GetSupportedResourceTypes
	backup:GetTieringConfiguration
	backup:ListBackupJobSummaries
	backup:ListBackupJobs
	backup:ListBackupPlanTemplates
	backup:ListBackupPlanVersions
	backup:ListBackupPlans
	backup:ListBackupSelections
	backup:ListBackupVaults
	backup:ListCopyJobSummaries
	backup:ListCopyJobs
	backup:ListFrameworks

Service prefix	Actions
	backup:ListIndexedRecoveryPoints
	backup:ListLegalHolds
	backup:ListProtectedResources
	backup:ListRecoveryPointsByBackupVault
	backup:ListRecoveryPointsByLegalHold
	backup:ListRecoveryPointsByResource
	backup:ListReportJobs
	backup:ListReportPlans
	backup:ListRestoreAccessBackupVaults
	backup:ListRestoreJobSummaries
	backup:ListRestoreJobs
	backup:ListRestoreJobsByProtectedResource
	backup:ListRestoreTestingPlans
	backup:ListRestoreTestingSelections
	backup:ListScanJobSummaries
	backup:ListScanJobs
	backup:ListTieringConfigurations
	backup:PutBackupVaultAccessPolicy
	backup:PutBackupVaultLockConfiguration
	backup:PutBackupVaultNotifications
	backup:PutRestoreValidationResult

Service prefix	Actions
	backup:StartBackupJob
	backup:StartCopyJob
	backup:StartReportJob
	backup:StartRestoreJob
	backup:StopBackupJob
	backup:UpdateBackupPlan
	backup:UpdateFramework
	backup:UpdateGlobalSettings
	backup:UpdateRecoveryPointLifecycle
	backup:UpdateRegionSettings
	backup:UpdateReportPlan
	backup:UpdateRestoreTestingPlan
	backup:UpdateRestoreTestingSelection
	backup:UpdateTieringConfiguration

Service prefix	Actions
batch	batch:CancelJob batch:CreateComputeEnvironment batch:CreateConsumableResource batch:CreateJobQueue batch:CreateQuotaShare batch:CreateSchedulingPolicy batch:CreateServiceEnvironment batch:DeleteComputeEnvironment batch:DeleteConsumableResource batch:DeleteJobQueue batch:DeleteQuotaShare batch:DeleteSchedulingPolicy batch:DeleteServiceEnvironment batch:DeregisterJobDefinition batch:DescribeComputeEnvironments batch:DescribeConsumableResource batch:DescribeJobDefinitions batch:DescribeJobQueues batch:DescribeJobs batch:DescribeQuotaShare batch:DescribeSchedulingPolicies

Service prefix	Actions
	batch:DescribeServiceEnvironments batch:DescribeServiceJob batch:GetJobQueueSnapshot batch:ListConsumableResources batch:ListJobs batch:ListJobsByConsumableResource batch:ListQuotaShares batch:ListSchedulingPolicies batch:ListServiceJobs batch:RegisterJobDefinition batch:SubmitJob batch:SubmitServiceJob batch:TerminateJob batch:TerminateServiceJob batch:UpdateComputeEnvironment batch:UpdateConsumableResource batch:UpdateJobQueue batch:UpdateQuotaShare batch:UpdateSchedulingPolicy batch:UpdateServiceEnvironment batch:UpdateServiceJob

Service prefix	Actions
braket	braket:CancelJob braket:CancelQuantumTask braket:CreateJob braket:CreateQuantumTask braket:CreateSpendingLimit braket:GetDevice braket:GetJob braket:GetQuantumTask braket:SearchDevices braket:SearchJobs braket:SearchQuantumTasks braket:SearchSpendingLimits

Service prefix	Actions
budgets	budgets:CreateBudgetAction budgets:DeleteBudgetAction budgets:DescribeBudgetAction budgets:DescribeBudgetActionHistories budgets:DescribeBudgetActionsForAccount budgets:DescribeBudgetActionsForBudget budgets:ExecuteBudgetAction budgets:ModifyBudget budgets:UpdateBudgetAction budgets:ViewBudget
cloud9	cloud9:CreateEnvironmentEC2 cloud9:CreateEnvironmentMembership cloud9:DeleteEnvironment cloud9:DeleteEnvironmentMembership cloud9:DescribeEnvironmentMemberships cloud9:DescribeEnvironmentStatus cloud9:DescribeEnvironments cloud9:ListEnvironments cloud9:UpdateEnvironment cloud9:UpdateEnvironmentMembership

Service prefix	Actions
cloudformation	cloudformation:BatchDescribeTypeConfigurations cloudformation:CancelUpdateStack cloudformation:ContinueUpdateRollback cloudformation:CreateChangeSet cloudformation:CreateGeneratedTemplate cloudformation:CreateStack cloudformation:CreateStackInstances cloudformation:CreateStackSet cloudformation:DeactivateType cloudformation:DeleteChangeSet cloudformation:DeleteGeneratedTemplate cloudformation:DeleteStack cloudformation:DeleteStackInstances cloudformation:DeleteStackSet cloudformation:DeregisterType cloudformation:DescribeAccountLimits cloudformation:DescribeChangeSet cloudformation:DescribeChangeSetHooks cloudformation:DescribeEvents cloudformation:DescribeGeneratedTemplate cloudformation:DescribeOrganizationsAccess

Service prefix	Actions
	cloudformation:DescribePublisher cloudformation:DescribeResourceScan cloudformation:DescribeStackDriftDetectionStatus cloudformation:DescribeStackEvents cloudformation:DescribeStackInstance cloudformation:DescribeStackResource cloudformation:DescribeStackResourceDrifts cloudformation:DescribeStackResources cloudformation:DescribeStackSet cloudformation:DescribeStackSetOperation cloudformation:DescribeStacks cloudformation:DescribeType cloudformation:DescribeTypeRegistration cloudformation:DetectStackDrift cloudformation:DetectStackResourceDrift cloudformation:DetectStackSetDrift cloudformation:EstimateTemplateCost cloudformation:ExecuteChangeSet cloudformation:GetGeneratedTemplate cloudformation:GetHookResult cloudformation:GetStackPolicy

Service prefix	Actions
	cloudformation:GetTemplate cloudformation:GetTemplateSummary cloudformation:ImportStacksToStackSet cloudformation:ListChangeSets cloudformation:ListExports cloudformation:ListGeneratedTemplates cloudformation:ListHookResults cloudformation:ListImports cloudformation:ListResourceScanRelatedResources cloudformation:ListResourceScanResources cloudformation:ListResourceScans cloudformation:ListStackInstanceResourceDrifts cloudformation:ListStackInstances cloudformation:ListStackRefactors cloudformation:ListStackResources cloudformation:ListStackSetAutoDeploymentTargets cloudformation:ListStackSetOperationResults cloudformation:ListStackSetOperations cloudformation:ListStackSets cloudformation:ListTypeRegistrations cloudformation:ListTypeVersions

Service prefix	Actions
	cloudformation:ListTypes cloudformation:PublishType cloudformation:RecordHandlerProgress cloudformation:RegisterPublisher cloudformation:RegisterType cloudformation:RollbackStack cloudformation:SetStackPolicy cloudformation:SetTypeConfiguration cloudformation:SetTypeDefaultVersion cloudformation:SignalResource cloudformation:StartResourceScan cloudformation:StopStackSetOperation cloudformation:TestType cloudformation:UpdateGeneratedTemplate cloudformation:UpdateStack cloudformation:UpdateStackInstances cloudformation:UpdateStackSet cloudformation:UpdateTerminationProtection cloudformation:ValidateTemplate

Service prefix	Actions
cloudfront	cloudfront:AssociateAlias
	cloudfront:AssociateDistributionTenantWebACL
	cloudfront:AssociateDistributionWebACL
	cloudfront:CreateCachePolicy
	cloudfront:CreateCloudFrontOriginAccessIdentity
	cloudfront:CreateConnectionFunction
	cloudfront:CreateContinuousDeploymentPolicy
	cloudfront:CreateDistributionTenant
	cloudfront:CreateFieldLevelEncryptionConfig
	cloudfront:CreateFieldLevelEncryptionProfile
	cloudfront:CreateFunction
	cloudfront:CreateInvalidation
	cloudfront:CreateKeyGroup
	cloudfront:CreateKeyValueStore
	cloudfront:CreateMonitoringSubscription
	cloudfront:CreateOriginAccessControl
	cloudfront:CreateOriginRequestPolicy
	cloudfront:CreatePublicKey
	cloudfront:CreateRealtimeLogConfig
	cloudfront:CreateResponseHeadersPolicy
	cloudfront:CreateTrustStore

Service prefix	Actions
	cloudfront:DeleteAnycastIpList
	cloudfront:DeleteCachePolicy
	cloudfront:DeleteCloudFrontOriginAccessIdentity
	cloudfront:DeleteConnectionFunction
	cloudfront:DeleteConnectionGroup
	cloudfront:DeleteContinuousDeploymentPolicy
	cloudfront:DeleteDistribution
	cloudfront:DeleteDistributionTenant
	cloudfront:DeleteFieldLevelEncryptionConfig
	cloudfront:DeleteFieldLevelEncryptionProfile
	cloudfront:DeleteFunction
	cloudfront:DeleteKeyGroup
	cloudfront:DeleteKeyValueStore
	cloudfront:DeleteMonitoringSubscription
	cloudfront:DeleteOriginAccessControl
	cloudfront:DeleteOriginRequestPolicy
	cloudfront:DeletePublicKey
	cloudfront:DeleteRealtimeLogConfig
	cloudfront:DeleteResponseHeadersPolicy
	cloudfront:DeleteStreamingDistribution
	cloudfront:DeleteTrustStore

Service prefix	Actions
	cloudfront:DeleteVpcOrigin
	cloudfront:DescribeFunction
	cloudfront:DescribeKeyValueStore
	cloudfront:DisassociateDistributionTenantWebACL
	cloudfront:DisassociateDistributionWebACL
	cloudfront:GetAnycastIpList
	cloudfront:GetCachePolicy
	cloudfront:GetCachePolicyConfig
	cloudfront:GetCloudFrontOriginAccessIdentity
	cloudfront:GetCloudFrontOriginAccessIdentityConfig
	cloudfront:GetContinuousDeploymentPolicy
	cloudfront:GetContinuousDeploymentPolicyConfig
	cloudfront:GetDistributionConfig
	cloudfront:GetFieldLevelEncryption
	cloudfront:GetFieldLevelEncryptionConfig
	cloudfront:GetFieldLevelEncryptionProfile
	cloudfront:GetFieldLevelEncryptionProfileConfig
	cloudfront:GetFunction
	cloudfront:GetInvalidation
	cloudfront:GetInvalidationForDistributionTenant
	cloudfront:GetKeyGroup

Service prefix	Actions
	cloudfront:GetKeyGroupConfig
	cloudfront:GetMonitoringSubscription
	cloudfront:GetOriginAccessControl
	cloudfront:GetOriginAccessControlConfig
	cloudfront:GetOriginRequestPolicy
	cloudfront:GetOriginRequestPolicyConfig
	cloudfront:GetPublicKey
	cloudfront:GetPublicKeyConfig
	cloudfront:GetRealtimeLogConfig
	cloudfront:GetResponseHeadersPolicy
	cloudfront:GetResponseHeadersPolicyConfig
	cloudfront:GetStreamingDistribution
	cloudfront:GetStreamingDistributionConfig
	cloudfront:GetVpcOrigin
	cloudfront:ListAnycastIpLists
	cloudfront:ListCachePolicies
	cloudfront:ListCloudFrontOriginAccessIdentities
	cloudfront:ListConflictingAliases
	cloudfront:ListConnectionFunctions
	cloudfront:ListConnectionGroups
	cloudfront:ListContinuousDeploymentPolicies

Service prefix	Actions
	cloudfront:ListDistributionTenants
	cloudfront:ListDistributionTenantsByCustomization
	cloudfront:ListDistributions
	cloudfront:ListDistributionsByAnycastIpListId
	cloudfront:ListDistributionsByCachePolicyId
	cloudfront:ListDistributionsByConnectionMode
	cloudfront:ListDistributionsByKeyGroup
	cloudfront:ListDistributionsByOriginRequestPolicyId
	cloudfront:ListDistributionsByRealtimeLogConfig
	cloudfront:ListDistributionsByResponseHeadersPolicyId
	cloudfront:ListDistributionsByVpcOriginId
	cloudfront:ListDistributionsByWebACLId
	cloudfront:ListFieldLevelEncryptionConfigs
	cloudfront:ListFieldLevelEncryptionProfiles
	cloudfront:ListFunctions
	cloudfront:ListInvalidations
	cloudfront:ListInvalidationsForDistributionTenant
	cloudfront:ListKeyGroups
	cloudfront:ListKeyValueStores
	cloudfront:ListOriginAccessControls
	cloudfront:ListOriginRequestPolicies

Service prefix	Actions
	cloudfront:ListPublicKeys
	cloudfront:ListRealtimeLogConfigs
	cloudfront:ListResponseHeadersPolicies
	cloudfront:ListStreamingDistributions
	cloudfront:ListTrustStores
	cloudfront:PublishConnectionFunction
	cloudfront:PublishFunction
	cloudfront:TestConnectionFunction
	cloudfront:TestFunction
	cloudfront:UpdateAnycastIpList
	cloudfront:UpdateCachePolicy
	cloudfront:UpdateCloudFrontOriginAccessIdentity
	cloudfront:UpdateConnectionFunction
	cloudfront:UpdateConnectionGroup
	cloudfront:UpdateContinuousDeploymentPolicy
	cloudfront:UpdateDistribution
	cloudfront:UpdateDistributionTenant
	cloudfront:UpdateFieldLevelEncryptionConfig
	cloudfront:UpdateFieldLevelEncryptionProfile
	cloudfront:UpdateFunction
	cloudfront:UpdateKeyGroup

Service prefix	Actions
	cloudfront:UpdateKeyValueStore cloudfront:UpdateOriginAccessControl cloudfront:UpdateOriginRequestPolicy cloudfront:UpdatePublicKey cloudfront:UpdateRealtimeLogConfig cloudfront:UpdateResponseHeadersPolicy cloudfront:UpdateTrustStore
cloudhsm	cloudhsm:CreateHsm cloudhsm:DeleteBackup cloudhsm:DeleteHsm cloudhsm:DeleteResourcePolicy cloudhsm:DescribeBackups cloudhsm:DescribeClusters cloudhsm:GetResourcePolicy cloudhsm:InitializeCluster cloudhsm:ModifyBackupAttributes cloudhsm:ModifyCluster cloudhsm:PutResourcePolicy cloudhsm:RestoreBackup

Service prefix	Actions
cloudsearch	cloudsearch:BuildSuggesters
	cloudsearch:CreateDomain
	cloudsearch:DefineAnalysisScheme
	cloudsearch:DefineExpression
	cloudsearch:DefineIndexField
	cloudsearch:DefineSuggester
	cloudsearch:DeleteAnalysisScheme
	cloudsearch:DeleteDomain
	cloudsearch:DeleteExpression
	cloudsearch:DeleteIndexField
	cloudsearch:DeleteSuggester
	cloudsearch:DescribeAnalysisSchemes
	cloudsearch:DescribeAvailabilityOptions
	cloudsearch:DescribeDomainEndpointOptions
	cloudsearch:DescribeDomains
	cloudsearch:DescribeExpressions
	cloudsearch:DescribeIndexFields
	cloudsearch:DescribeScalingParameters
	cloudsearch:DescribeServiceAccessPolicies
	cloudsearch:DescribeSuggesters
	cloudsearch:IndexDocuments

Service prefix	Actions
	cloudsearch:ListDomainNames cloudsearch:UpdateAvailabilityOptions cloudsearch:UpdateDomainEndpointOptions cloudsearch:UpdateScalingParameters cloudsearch:UpdateServiceAccessPolicies

Service prefix	Actions
cloudtrail	cloudtrail:CancelQuery cloudtrail:CreateChannel cloudtrail:CreateDashboard cloudtrail:CreateEventDataStore cloudtrail:CreateTrail cloudtrail:DeleteChannel cloudtrail:DeleteDashboard cloudtrail:DeleteEventDataStore cloudtrail:DeleteResourcePolicy cloudtrail:DeleteTrail cloudtrail:DeregisterOrganizationDelegatedAdmin cloudtrail:DescribeQuery cloudtrail:DescribeTrails cloudtrail:DisableFederation cloudtrail:GenerateQuery cloudtrail:GetChannel cloudtrail:GetDashboard cloudtrail:GetEventConfiguration cloudtrail:GetEventDataStore cloudtrail:GetEventDataStoreData cloudtrail:GetEventSelectors

Service prefix	Actions
	cloudtrail:GetImport cloudtrail:GetInsightSelectors cloudtrail:GetResourcePolicy cloudtrail:GetTrail cloudtrail:GetTrailStatus cloudtrail:ListChannels cloudtrail:ListDashboards cloudtrail:ListEventDataStores cloudtrail:ListImportFailures cloudtrail:ListImports cloudtrail:ListInsightsData cloudtrail:ListPublicKeys cloudtrail:ListQueries cloudtrail:ListTrails cloudtrail:LookupEvents cloudtrail:PutEventConfiguration cloudtrail:PutEventSelectors cloudtrail:PutInsightSelectors cloudtrail:PutResourcePolicy cloudtrail:RegisterOrganizationDelegatedAdmin cloudtrail:RestoreEventDataStore

Service prefix	Actions
	cloudtrail:SearchSampleQueries cloudtrail:StartEventDataStoreIngestion cloudtrail:StartImport cloudtrail:StartLogging cloudtrail:StartQuery cloudtrail:StopEventDataStoreIngestion cloudtrail:StopImport cloudtrail:StopLogging cloudtrail:UpdateChannel cloudtrail:UpdateDashboard cloudtrail:UpdateEventDataStore cloudtrail:UpdateTrail

Service prefix	Actions
cloudwatch	cloudwatch:DeleteAlarmMuteRule
	cloudwatch:DeleteAlarms
	cloudwatch:DeleteAnomalyDetector
	cloudwatch:DeleteDashboards
	cloudwatch:DeleteInsightRules
	cloudwatch:DeleteMetricStream
	cloudwatch:DescribeAlarmHistory
	cloudwatch:DescribeAlarms
	cloudwatch:DescribeAlarmsForMetric
	cloudwatch:DescribeAnomalyDetectors
	cloudwatch:DescribeInsightRules
	cloudwatch:DisableAlarmActions
	cloudwatch:DisableInsightRules
	cloudwatch:EnableAlarmActions
	cloudwatch:EnableInsightRules
	cloudwatch:GetAlarmMuteRule
	cloudwatch:GetDashboard
	cloudwatch:GetDataset
	cloudwatch:GetInsightRuleReport
	cloudwatch:GetMetricStatistics
	cloudwatch:GetMetricStream

Service prefix	Actions
	cloudwatch:ListAlarmMuteRules
	cloudwatch:ListDashboards
	cloudwatch:ListManagedInsightRules
	cloudwatch:ListMetricStreams
	cloudwatch:PutAlarmMuteRule
	cloudwatch:PutAnomalyDetector
	cloudwatch:PutCompositeAlarm
	cloudwatch:PutDashboard
	cloudwatch:PutInsightRule
	cloudwatch:PutLogAlarm
	cloudwatch:PutManagedInsightRules
	cloudwatch:PutMetricAlarm
	cloudwatch:PutMetricStream
	cloudwatch:SetAlarmState
	cloudwatch:StartMetricStreams
	cloudwatch:StopMetricStreams

Service prefix	Actions
codeartifact	codeartifact:AssociateExternalConnection codeartifact:CopyPackageVersions codeartifact:CreateDomain codeartifact:CreateRepository codeartifact:DeleteDomain codeartifact:DeleteDomainPermissionsPolicy codeartifact:DeletePackage codeartifact:DeletePackageVersions codeartifact:DeleteRepository codeartifact:DeleteRepositoryPermissionsPolicy codeartifact:DescribeDomain codeartifact:DescribePackage codeartifact:DescribePackageVersion codeartifact:DescribeRepository codeartifact:DisassociateExternalConnection codeartifact:DisposePackageVersions codeartifact:GetAssociatedPackageGroup codeartifact:GetAuthorizationToken codeartifact:GetDomainPermissionsPolicy codeartifact:GetPackageVersionAsset codeartifact:GetPackageVersionReadme

Service prefix	Actions
	codeartifact:GetRepositoryEndpoint codeartifact:GetRepositoryPermissionsPolicy codeartifact:ListDomains codeartifact:ListPackageGroups codeartifact:ListPackageVersionAssets codeartifact:ListPackageVersionDependencies codeartifact:ListPackageVersions codeartifact:ListPackages codeartifact:ListRepositories codeartifact:ListRepositoriesInDomain codeartifact:PublishPackageVersion codeartifact:PutDomainPermissionsPolicy codeartifact:PutPackageMetadata codeartifact:PutPackageOriginConfiguration codeartifact:PutRepositoryPermissionsPolicy codeartifact:ReadFromRepository codeartifact:UpdatePackageVersionsStatus codeartifact:UpdateRepository

Service prefix	Actions
codedeploy	codedeploy:BatchGetApplicationRevisions
	codedeploy:BatchGetApplications
	codedeploy:BatchGetDeploymentGroups
	codedeploy:BatchGetDeploymentInstances
	codedeploy:BatchGetDeploymentTargets
	codedeploy:BatchGetDeployments
	codedeploy:BatchGetOnPremisesInstances
	codedeploy:ContinueDeployment
	codedeploy:CreateApplication
	codedeploy:CreateDeployment
	codedeploy:CreateDeploymentConfig
	codedeploy:CreateDeploymentGroup
	codedeploy:DeleteApplication
	codedeploy:DeleteDeploymentConfig
	codedeploy:DeleteDeploymentGroup
	codedeploy:DeleteGitHubAccountToken
	codedeploy:DeleteResourcesByExternalId
	codedeploy:DeregisterOnPremisesInstance
	codedeploy:GetApplication
	codedeploy:GetApplicationRevision
	codedeploy:GetDeployment

Service prefix	Actions
	codedeploy:GetDeploymentConfig
	codedeploy:GetDeploymentGroup
	codedeploy:GetDeploymentInstance
	codedeploy:GetDeploymentTarget
	codedeploy:GetOnPremisesInstance
	codedeploy:ListApplicationRevisions
	codedeploy:ListApplications
	codedeploy:ListDeploymentConfigs
	codedeploy:ListDeploymentGroups
	codedeploy:ListDeploymentInstances
	codedeploy:ListDeploymentTargets
	codedeploy:ListDeployments
	codedeploy:ListGitHubAccountTokenNames
	codedeploy:ListOnPremisesInstances
	codedeploy:PutLifecycleEventHookExecutionStatus
	codedeploy:RegisterApplicationRevision
	codedeploy:RegisterOnPremisesInstance
	codedeploy:SkipWaitTimeForInstanceTermination
	codedeploy:StopDeployment
	codedeploy:UpdateApplication
	codedeploy:UpdateDeploymentGroup

Service prefix	Actions
codeguru-profiler	codeguru-profiler:AddNotificationChannels codeguru-profiler:BatchGetFrameMetricData codeguru-profiler:CreateProfilingGroup codeguru-profiler:DeleteProfilingGroup codeguru-profiler:DescribeProfilingGroup codeguru-profiler:GetFindingsReportAccountSummary codeguru-profiler:GetNotificationConfiguration codeguru-profiler:GetPolicy codeguru-profiler:GetProfile codeguru-profiler:GetRecommendations codeguru-profiler:ListFindingsReports codeguru-profiler:ListProfileTimes codeguru-profiler:ListProfilingGroups codeguru-profiler:PutPermission codeguru-profiler:RemoveNotificationChannel codeguru-profiler:RemovePermission codeguru-profiler:SubmitFeedback codeguru-profiler:UpdateProfilingGroup

Service prefix	Actions
codeguru-reviewer	codeguru-reviewer:AssociateRepository codeguru-reviewer:CreateCodeReview codeguru-reviewer:DescribeCodeReview codeguru-reviewer:DescribeRecommendationFeedback codeguru-reviewer:DescribeRepositoryAssociation codeguru-reviewer:DisassociateRepository codeguru-reviewer:ListCodeReviews codeguru-reviewer:ListRecommendationFeedback codeguru-reviewer:ListRecommendations codeguru-reviewer:ListRepositoryAssociations codeguru-reviewer:PutRecommendationFeedback

Service prefix	Actions
codepipeline	codepipeline:AcknowledgeJob codepipeline:AcknowledgeThirdPartyJob codepipeline:CreateCustomActionType codepipeline:CreatePipeline codepipeline:DeleteCustomActionType codepipeline:DeletePipeline codepipeline:DeleteWebhook codepipeline:DeregisterWebhookWithThirdParty codepipeline:GetActionType codepipeline:GetJobDetails codepipeline:GetPipeline codepipeline:GetPipelineExecution codepipeline:GetPipelineState codepipeline:GetThirdPartyJobDetails codepipeline:ListActionExecutions codepipeline:ListActionTypes codepipeline:ListPipelineExecutions codepipeline:ListPipelines codepipeline:ListRuleExecutions codepipeline:ListRuleTypes codepipeline:ListWebhooks

Service prefix	Actions
	codepipeline:OverrideStageCondition codepipeline:PollForJobs codepipeline:PollForThirdPartyJobs codepipeline:PutActionRevision codepipeline:PutApprovalResult codepipeline:PutJobFailureResult codepipeline:PutJobSuccessResult codepipeline:PutThirdPartyJobFailureResult codepipeline:PutThirdPartyJobSuccessResult codepipeline:PutWebhook codepipeline:RegisterWebhookWithThirdParty codepipeline:RollbackStage codepipeline:StartPipelineExecution codepipeline:StopPipelineExecution codepipeline:UpdateActionType codepipeline:UpdatePipeline

Service prefix	Actions
codestar	codestar:AssociateTeamMember
	codestar:CreateProject
	codestar:CreateUserProfile
	codestar:DeleteProject
	codestar:DeleteUserProfile
	codestar:DescribeProject
	codestar:DescribeUserProfile
	codestar:DisassociateTeamMember
	codestar:ListProjects
	codestar:ListResources
	codestar:ListTeamMembers
	codestar:ListUserProfiles
	codestar:UpdateProject
	codestar:UpdateTeamMember
	codestar:UpdateUserProfile

Service prefix	Actions
codestar-notifications	codestar-notifications:CreateNotificationRule codestar-notifications:DeleteNotificationRule codestar-notifications:DeleteTarget codestar-notifications:DescribeNotificationRule codestar-notifications:ListEventTypes codestar-notifications:ListNotificationRules codestar-notifications:ListTargets codestar-notifications:Subscribe codestar-notifications:Unsubscribe codestar-notifications:UpdateNotificationRule

Service prefix	Actions
cognito-identity	cognito-identity:CreateIdentityPool
	cognito-identity:DeleteIdentities
	cognito-identity:DeleteIdentityPool
	cognito-identity:DescribeIdentity
	cognito-identity:DescribeIdentityPool
	cognito-identity:GetIdentityPoolRoles
	cognito-identity:ListIdentities
	cognito-identity:ListIdentityPools
	cognito-identity:LookupDeveloperIdentity
	cognito-identity:MergeDeveloperIdentities
	cognito-identity:SetIdentityPoolRoles
	cognito-identity:UnlinkDeveloperIdentity
	cognito-identity:UpdateIdentityPool

Service prefix	Actions
cognito-idp	cognito-idp:AddCustomAttributes cognito-idp:AddUserPoolClientSecret cognito-idp:AdminAddUserToGroup cognito-idp:AdminConfirmSignUp cognito-idp:AdminCreateUser cognito-idp:AdminDeleteUser cognito-idp:AdminDeleteUserAttributes cognito-idp:AdminDisableProviderForUser cognito-idp:AdminDisableUser cognito-idp:AdminEnableUser cognito-idp:AdminForgetDevice cognito-idp:AdminGetDevice cognito-idp:AdminGetUser cognito-idp:AdminGetUserAuthFactors cognito-idp:AdminInitiateAuth cognito-idp:AdminLinkProviderForUser cognito-idp:AdminListDevices cognito-idp:AdminListGroupsForUser cognito-idp:AdminListUserAuthEvents cognito-idp:AdminRemoveUserFromGroup cognito-idp:AdminResetUserPassword

Service prefix	Actions
	cognito-idp:AdminRespondToAuthChallenge cognito-idp:AdminSetUserMFAPreference cognito-idp:AdminSetUserPassword cognito-idp:AdminSetUserSettings cognito-idp:AdminUpdateAuthEventFeedback cognito-idp:AdminUpdateDeviceStatus cognito-idp:AdminUpdateUserAttributes cognito-idp:AdminUserGlobalSignOut cognito-idp:AssociateSoftwareToken cognito-idp:ChangePassword cognito-idp:ConfirmDevice cognito-idp:ConfirmForgotPassword cognito-idp:ConfirmSignUp cognito-idp:CreateGroup cognito-idp:CreateIdentityProvider cognito-idp:CreateManagedLoginBranding cognito-idp:CreateResourceServer cognito-idp:CreateTerms cognito-idp:CreateUserImportJob cognito-idp:CreateUserPool cognito-idp:CreateUserPoolClient

Service prefix	Actions
	cognito-idp:CreateUserPoolDomain cognito-idp:CreateUserPoolReplica cognito-idp:DeleteGroup cognito-idp:DeleteIdentityProvider cognito-idp:DeleteManagedLoginBranding cognito-idp:DeleteResourceServer cognito-idp:DeleteTerms cognito-idp:DeleteUser cognito-idp:DeleteUserAttributes cognito-idp:DeleteUserPool cognito-idp:DeleteUserPoolClient cognito-idp:DeleteUserPoolClientSecret cognito-idp:DeleteUserPoolDomain cognito-idp:DeleteUserPoolReplica cognito-idp:DescribeIdentityProvider cognito-idp:DescribeManagedLoginBranding cognito-idp:DescribeManagedLoginBrandingByClient cognito-idp:DescribeResourceServer cognito-idp:DescribeRiskConfiguration cognito-idp:DescribeTerms cognito-idp:DescribeUserImportJob

Service prefix	Actions
	cognito-idp:DescribeUserPool cognito-idp:DescribeUserPoolClient cognito-idp:DescribeUserPoolDomain cognito-idp:ForgetDevice cognito-idp:ForgotPassword cognito-idp:GetCSVHeader cognito-idp:GetDevice cognito-idp:GetGroup cognito-idp:GetIdentityProviderByIdentifier cognito-idp:GetLogDeliveryConfiguration cognito-idp:GetProvisionedLimit cognito-idp:GetSigningCertificate cognito-idp:GetUICustomization cognito-idp:GetUser cognito-idp:GetUserAttributeVerificationCode cognito-idp:GetUserPoolMfaConfig cognito-idp:GlobalSignOut cognito-idp:InitiateAuth cognito-idp:ListDevices cognito-idp:ListGroups cognito-idp:ListIdentityProviders

Service prefix	Actions
	cognito-idp:ListResourceServers cognito-idp:ListTerms cognito-idp:ListUserImportJobs cognito-idp:ListUserPoolClientSecrets cognito-idp:ListUserPoolClients cognito-idp:ListUserPoolReplicas cognito-idp:ListUserPools cognito-idp:ListUsers cognito-idp:ListUsersInGroup cognito-idp:ResendConfirmationCode cognito-idp:RespondToAuthChallenge cognito-idp:RevokeToken cognito-idp:SetLogDeliveryConfiguration cognito-idp:SetRiskConfiguration cognito-idp:SetUICustomization cognito-idp:SetUserMFAPreference cognito-idp:SetUserPoolMfaConfig cognito-idp:SetUserSettings cognito-idp:SignUp cognito-idp:StartUserImportJob cognito-idp:StopUserImportJob

Service prefix	Actions
	cognito-idp:UpdateAuthEventFeedback cognito-idp:UpdateDeviceStatus cognito-idp:UpdateGroup cognito-idp:UpdateIdentityProvider cognito-idp:UpdateProvisionedLimit cognito-idp:UpdateResourceServer cognito-idp:UpdateTerms cognito-idp:UpdateUserAttributes cognito-idp:UpdateUserPool cognito-idp:UpdateUserPoolClient cognito-idp:UpdateUserPoolDomain cognito-idp:UpdateUserPoolReplica cognito-idp:VerifySoftwareToken cognito-idp:VerifyUserAttribute

Service prefix	Actions
cognito-sync	cognito-sync:BulkPublish
	cognito-sync:DeleteDataset
	cognito-sync:DescribeDataset
	cognito-sync:DescribeIdentityPoolUsage
	cognito-sync:DescribeIdentityUsage
	cognito-sync:GetBulkPublishDetails
	cognito-sync:GetCognitoEvents
	cognito-sync:GetIdentityPoolConfiguration
	cognito-sync:ListDatasets
	cognito-sync:ListIdentityPoolUsage
	cognito-sync:ListRecords
	cognito-sync:RegisterDevice
	cognito-sync:SetCognitoEvents
	cognito-sync:SetIdentityPoolConfiguration
	cognito-sync:SubscribeToDataset
	cognito-sync:UnsubscribeFromDataset
	cognito-sync:UpdateRecords

Service prefix	Actions
comprehendmedical	comprehendmedical:DescribeEntitiesDetectionV2Job comprehendmedical:DescribeICD10CMInferenceJob comprehendmedical:DescribePHIDetectionJob comprehendmedical:DescribeRxNormInferenceJob comprehendmedical:DescribeSNOMEDCTInferenceJob comprehendmedical:DetectEntitiesV2 comprehendmedical:DetectPHI comprehendmedical:InferICD10CM comprehendmedical:InferRxNorm comprehendmedical:InferSNOMEDCT comprehendmedical:ListEntitiesDetectionV2Jobs comprehendmedical:ListICD10CMInferenceJobs comprehendmedical:ListPHIDetectionJobs comprehendmedical:ListRxNormInferenceJobs comprehendmedical:ListSNOMEDCTInferenceJobs comprehendmedical:StartEntitiesDetectionV2Job comprehendmedical:StartICD10CMInferenceJob comprehendmedical:StartPHIDetectionJob comprehendmedical:StartRxNormInferenceJob comprehendmedical:StartSNOMEDCTInferenceJob comprehendmedical:StopEntitiesDetectionV2Job

Service prefix	Actions
	comprehendmedical:StopICD10CMInferenceJob comprehendmedical:StopPHIDetectionJob comprehendmedical:StopRxNormInferenceJob comprehendmedical:StopSNOMEDCTInferenceJob

Service prefix	Actions
compute-optimizer	compute-optimizer:DeleteRecommendationPreferences
	compute-optimizer:DescribeRecommendationExportJobs
	compute-optimizer:ExportAutoScalingGroupRecommendations
	compute-optimizer:ExportEBSVolumeRecommendations
	compute-optimizer:ExportEC2InstanceRecommendations
	compute-optimizer:ExportECSServiceRecommendations
	compute-optimizer:ExportIdleRecommendations
	compute-optimizer:ExportLambdaFunctionRecommendations
	compute-optimizer:ExportLicenseRecommendations
	compute-optimizer:ExportRDSDatabaseRecommendations
	compute-optimizer:GetEC2RecommendationProjectedMetrics
	compute-optimizer:GetECSServiceRecommendationProjectedMetrics
	compute-optimizer:GetEffectiveRecommendationPreferences
	compute-optimizer:GetEnrollmentStatus
	compute-optimizer:GetEnrollmentStatusesForOrganization
	compute-optimizer:GetRDSDatabaseRecommendationProjectedMetrics
	compute-optimizer:GetRecommendationPreferences
	compute-optimizer:GetRecommendationSummaries
	compute-optimizer:PutRecommendationPreferences

Service prefix	Actions
	compute-optimizer:UpdateEnrollmentStatus

Service prefix	Actions
config	config:BatchGetResourceConfig config:DeleteAggregationAuthorization config:DeleteConfigRule config:DeleteConfigurationAggregator config:DeleteConfigurationRecorder config:DeleteConformancePack config:DeleteDeliveryChannel config:DeleteEvaluationResults config:DeleteOrganizationConfigRule config:DeleteOrganizationConformancePack config:DeletePendingAggregationRequest config:DeleteRemediationConfiguration config:DeleteRemediationExceptions config:DeleteResourceConfig config:DeleteRetentionConfiguration config:DeleteStoredQuery config:DeliverConfigSnapshot config:DescribeAggregateComplianceByConfigRules config:DescribeAggregateComplianceByConformancePacks config:DescribeAggregationAuthorizations config:DescribeComplianceByConfigRule

Service prefix	Actions
	config:DescribeComplianceByResource config:DescribeConfigRuleEvaluationStatus config:DescribeConfigRules config:DescribeConfigurationAggregatorSourcesStatus config:DescribeConfigurationAggregators config:DescribeConfigurationRecorderStatus config:DescribeConfigurationRecorders config:DescribeConformancePackCompliance config:DescribeConformancePackStatus config:DescribeConformancePacks config:DescribeDeliveryChannelStatus config:DescribeDeliveryChannels config:DescribeOrganizationConfigRuleStatuses config:DescribeOrganizationConfigRules config:DescribeOrganizationConformancePackStatuses config:DescribeOrganizationConformancePacks config:DescribePendingAggregationRequests config:DescribeRemediationConfigurations config:DescribeRemediationExceptions config:DescribeRemediationExecutionStatus config:DescribeRetentionConfigurations

Service prefix	Actions
	config:GetComplianceDetailsByConfigRule config:GetComplianceDetailsByResource config:GetComplianceSummaryByConfigRule config:GetComplianceSummaryByResourceType config:GetConformancePackComplianceDetails config:GetConformancePackComplianceSummary config:GetCustomRulePolicy config:GetDiscoveredResourceCounts config:GetOrganizationConfigRuleDetailedStatus config:GetOrganizationConformancePackDetailedStatus config:GetOrganizationCustomRulePolicy config:GetResourceConfigHistory config:GetResourceEvaluationSummary config:GetStoredQuery config:ListConfigurationRecorders config:ListConformancePackComplianceScores config:ListConnectors config:ListDiscoveredResources config:ListResourceEvaluations config:ListStoredQueries config:PutConfigRule

Service prefix	Actions
	config:PutConfigurationAggregator config:PutConfigurationRecorder config:PutConformancePack config:PutConnector config:PutDeliveryChannel config:PutEvaluations config:PutExternalEvaluation config:PutOrganizationConfigRule config:PutOrganizationConformancePack config:PutRemediationConfigurations config:PutRemediationExceptions config:PutResourceConfig config:PutRetentionConfiguration config:PutStoredQuery config:SelectResourceConfig config:StartConfigRulesEvaluation config:StartConfigurationRecorder config:StartRemediationExecution config:StartResourceEvaluation config:StopConfigurationRecorder

Service prefix	Actions
connect	connect:ActivateEvaluationForm
	connect:AssociateAnalyticsDataSet
	connect:AssociateApprovedOrigin
	connect:AssociateBot
	connect:AssociateContactWithUser
	connect:AssociateDefaultVocabulary
	connect:AssociateEmailAddressAlias
	connect:AssociateFlow
	connect:AssociateInstanceStorageConfig
	connect:AssociateLambdaFunction
	connect:AssociateLexBot
	connect:AssociatePhoneNumberContactFlow
	connect:AssociateQueueEmailAddresses
	connect:AssociateQueueQuickConnects
	connect:AssociateRoutingProfileQueues
	connect:AssociateSecurityKey
	connect:AssociateUserProficiencies
	connect:BatchAssociateAnalyticsDataSet
	connect:BatchCreateDataTableValue
	connect:BatchDeleteDataTableValue
	connect:BatchDescribeDataTableValue

Service prefix	Actions
	connect:BatchDisassociateAnalyticsDataSet
	connect:BatchGetFlowAssociation
	connect:BatchPutContact
	connect:BatchUpdateDataTableValue
	connect:ClaimPhoneNumber
	connect:CreateAgentStatus
	connect:CreateAttachedFile
	connect:CreateContact
	connect:CreateContactFlow
	connect:CreateContactFlowModule
	connect:CreateContactFlowModuleAlias
	connect:CreateContactFlowModuleVersion
	connect:CreateContactFlowVersion
	connect:CreateDataTable
	connect:CreateDataTableAttribute
	connect:CreateEmailAddress
	connect:CreateEvaluationForm
	connect:CreateHoursOfOperation
	connect:CreateInstance
	connect:CreateIntegrationAssociation
	connect:CreateParticipant

Service prefix	Actions
	connect:CreatePersistentContactAssociation
	connect:CreatePredefinedAttribute
	connect:CreatePrompt
	connect:CreatePushNotificationRegistration
	connect:CreateQueue
	connect:CreateQuickConnect
	connect:CreateRoutingProfile
	connect:CreateRule
	connect:CreateSecurityProfile
	connect:CreateTaskTemplate
	connect:CreateTrafficDistributionGroup
	connect:CreateUseCase
	connect:CreateUser
	connect:CreateUserHierarchyGroup
	connect:CreateView
	connect:CreateViewVersion
	connect:CreateVocabulary
	connect:CreateWorkspace
	connect:DeactivateEvaluationForm
	connect:DeleteContactEvaluation
	connect:DeleteContactFlow

Service prefix	Actions
	connect:DeleteContactFlowModule
	connect:DeleteContactFlowModuleAlias
	connect:DeleteContactFlowModuleVersion
	connect:DeleteContactFlowVersion
	connect:DeleteDataTable
	connect:DeleteDataTableAttribute
	connect:DeleteEmailAddress
	connect:DeleteEvaluationForm
	connect:DeleteHoursOfOperation
	connect:DeleteHoursOfOperationOverride
	connect:DeleteInstance
	connect:DeleteIntegrationAssociation
	connect:DeleteNotification
	connect:DeletePredefinedAttribute
	connect:DeletePrompt
	connect:DeletePushNotificationRegistration
	connect:DeleteQueue
	connect:DeleteQuickConnect
	connect:DeleteRoutingProfile
	connect:DeleteRule
	connect:DeleteSecurityProfile

Service prefix	Actions
	connect:DeleteTaskTemplate
	connect:DeleteTrafficDistributionGroup
	connect:DeleteUseCase
	connect:DeleteUser
	connect:DeleteUserHierarchyGroup
	connect:DeleteView
	connect:DeleteVocabulary
	connect:DeleteWorkspace
	connect:DeleteWorkspaceMedia
	connect:DescribeAttachedFilesConfiguration
	connect:DescribeAuthenticationProfile
	connect:DescribeContactFlowModuleAlias
	connect:DescribeDataTableAttribute
	connect:DescribeHoursOfOperationOverride
	connect:DescribeInstanceAttribute
	connect:DescribeInstanceStorageConfig
	connect:DescribeNotification
	connect:DescribePhoneNumber
	connect:DescribeRule
	connect:DescribeTrafficDistributionGroup
	connect:DescribeUserHierarchyStructure

Service prefix	Actions
	connect:DescribeVocabulary
	connect:DisassociateAnalyticsDataSet
	connect:DisassociateApprovedOrigin
	connect:DisassociateBot
	connect:DisassociateEmailAddressAlias
	connect:DisassociateFlow
	connect:DisassociateInstanceStorageConfig
	connect:DisassociateLambdaFunction
	connect:DisassociateLexBot
	connect:DisassociatePhoneNumberContactFlow
	connect:DisassociateQueueEmailAddresses
	connect:DisassociateQueueQuickConnects
	connect:DisassociateRoutingProfileQueues
	connect:DisassociateSecurityKey
	connect:DisassociateUserProficiencies
	connect:DismissUserContact
	connect:EvaluateDataTableValues
	connect:GetContactAttributes
	connect:GetContactMetrics
	connect:GetCurrentMetricData
	connect:GetCurrentUserData

Service prefix	Actions
	connect:GetEffectiveHoursOfOperations
	connect:GetEvaluationFormValidation
	connect:GetFederationToken
	connect:GetFlowAssociation
	connect:GetMetricData
	connect:GetMetricDataV2
	connect:GetPromptFile
	connect:GetTaskTemplate
	connect:GetTrafficDistribution
	connect:ImportPhoneNumber
	connect:ImportWorkspaceMedia
	connect:ListAnalyticsDataAssociations
	connect:ListAnalyticsDataLakeDataSets
	connect:ListApprovedOrigins
	connect:ListAssociatedContacts
	connect:ListAttachedFilesConfigurations
	connect:ListAuthenticationProfiles
	connect:ListBots
	connect:ListContactEvaluations
	connect:ListContactFlowModuleAliases
	connect:ListContactFlowModuleVersions

Service prefix	Actions
	connect:ListContactFlowModules
	connect:ListContactFlowVersions
	connect:ListContactFlows
	connect:ListContactReferences
	connect:ListDataTableAttributes
	connect:ListDataTablePrimaryValues
	connect:ListDataTableValues
	connect:ListDataTables
	connect:ListDefaultVocabularies
	connect:ListEvaluationFormVersions
	connect:ListEvaluationForms
	connect:ListFlowAssociations
	connect:ListHoursOfOperations
	connect:ListInstanceAttributes
	connect:ListInstanceStorageConfigs
	connect:ListIntegrationAssociations
	connect:ListLambdaFunctions
	connect:ListLexBots
	connect:ListNotifications
	connect:ListPhoneNumbers
	connect:ListPhoneNumbersV2

Service prefix	Actions
	connect:ListPredefinedAttributes
	connect:ListPrompts
	connect:ListQueueEmailAddresses
	connect:ListQueueQuickConnects
	connect:ListQueues
	connect:ListQuickConnects
	connect:ListRealtimeContactAnalysisSegmentsV2
	connect:ListRoutingProfileManualAssignmentQueues
	connect:ListRoutingProfileQueues
	connect:ListRoutingProfiles
	connect:ListRules
	connect:ListSecurityKeys
	connect:ListSecurityProfileApplications
	connect:ListSecurityProfileFlowModules
	connect:ListSecurityProfilePermissions
	connect:ListSecurityProfiles
	connect:ListTaskTemplates
	connect:ListTrafficDistributionGroups
	connect:ListUseCases
	connect:ListUserHierarchyGroups
	connect:ListUserNotifications

Service prefix	Actions
	connect:ListUsers
	connect:ListViewVersions
	connect:ListViews
	connect:ListWorkspaceMedia
	connect:ListWorkspacePages
	connect:ListWorkspaces
	connect:MonitorContact
	connect:PauseContact
	connect:PutUserStatus
	connect:ReleasePhoneNumber
	connect:ReplicateInstance
	connect:ResumeContact
	connect:ResumeContactRecording
	connect:SearchAgentStatuses
	connect:SearchAvailablePhoneNumbers
	connect:SearchContactEvaluations
	connect:SearchContactFlowModules
	connect:SearchContactFlows
	connect:SearchContacts
	connect:SearchDataTables
	connect:SearchEmailAddresses

Service prefix	Actions
	connect:SearchEvaluationForms
	connect:SearchHoursOfOperations
	connect:SearchPredefinedAttributes
	connect:SearchPrompts
	connect:SearchQueues
	connect:SearchQuickConnects
	connect:SearchRoutingProfiles
	connect:SearchRules
	connect:SearchSecurityProfiles
	connect:SearchUserHierarchyGroups
	connect:SearchViews
	connect:SearchVocabularies
	connect:SearchWorkspaceAssociations
	connect:SearchWorkspaces
	connect:SendChatIntegrationEvent
	connect:SendOutboundEmail
	connect:StartChatContact
	connect:StartContactConversationalAnalyticsJob
	connect:StartContactEvaluation
	connect:StartContactMediaProcessing
	connect:StartContactRecording

Service prefix	Actions
	connect:StartContactStreaming
	connect:StartEmailContact
	connect:StartOutboundChatContact
	connect:StartOutboundEmailContact
	connect:StartOutboundVoiceContact
	connect:StartScreenSharing
	connect:StartTaskContact
	connect:StartWebRTCContact
	connect:StopContact
	connect:StopContactMediaProcessing
	connect:StopContactRecording
	connect:StopContactStreaming
	connect:SubmitContactEvaluation
	connect:SuspendContactRecording
	connect:TransferContact
	connect:UpdateAgentStatus
	connect:UpdateAttachedFilesConfiguration
	connect:UpdateAuthenticationProfile
	connect:UpdateContact
	connect:UpdateContactAttributes
	connect:UpdateContactEvaluation

Service prefix	Actions
	connect:UpdateContactFlowContent
	connect:UpdateContactFlowMetadata
	connect:UpdateContactFlowModuleAlias
	connect:UpdateContactFlowModuleContent
	connect:UpdateContactFlowModuleMetadata
	connect:UpdateContactFlowName
	connect:UpdateContactRoutingData
	connect:UpdateContactSchedule
	connect:UpdateDataTableAttribute
	connect:UpdateDataTableMetadata
	connect:UpdateDataTablePrimaryValues
	connect:UpdateEmailAddressMetadata
	connect:UpdateEvaluationForm
	connect:UpdateHoursOfOperation
	connect:UpdateHoursOfOperationOverride
	connect:UpdateInstanceAttribute
	connect:UpdateInstanceStorageConfig
	connect:UpdateNotificationContent
	connect:UpdateParticipantAuthentication
	connect:UpdateParticipantRoleConfig
	connect:UpdatePhoneNumber

Service prefix	Actions
	connect:UpdatePhoneNumberMetadata
	connect:UpdatePredefinedAttribute
	connect:UpdatePrompt
	connect:UpdateQueueHoursOfOperation
	connect:UpdateQueueMaxContacts
	connect:UpdateQueueName
	connect:UpdateQueueOutboundCallerConfig
	connect:UpdateQueueOutboundEmailConfig
	connect:UpdateQueueStatus
	connect:UpdateQuickConnectConfig
	connect:UpdateQuickConnectName
	connect:UpdateRoutingProfileAgentAvailabilityTimer
	connect:UpdateRoutingProfileConcurrency
	connect:UpdateRoutingProfileDefaultOutboundQueue
	connect:UpdateRoutingProfileName
	connect:UpdateRoutingProfileQueues
	connect:UpdateRule
	connect:UpdateSecurityProfile
	connect:UpdateTaskTemplate
	connect:UpdateTrafficDistribution
	connect:UpdateUserHierarchy

Service prefix	Actions
	connect:UpdateUserHierarchyGroupName connect:UpdateUserHierarchyStructure connect:UpdateUserIdentityInfo connect:UpdateUserNotificationStatus connect:UpdateUserPhoneConfig connect:UpdateUserProficiencies connect:UpdateUserRoutingProfile connect:UpdateUserSecurityProfiles connect:UpdateViewContent connect:UpdateViewMetadata connect:UpdateWorkspaceMetadata connect:UpdateWorkspaceTheme connect:UpdateWorkspaceVisibility
cur	cur:DeleteReportDefinition cur:DescribeReportDefinitions cur:ModifyReportDefinition cur:PutReportDefinition

Service prefix	Actions
databrew	databrew:BatchDeleteRecipeVersion
	databrew:CreateDataset
	databrew:CreateProfileJob
	databrew:CreateProject
	databrew:CreateRecipe
	databrew:CreateRecipeJob
	databrew:CreateRuleset
	databrew:CreateSchedule
	databrew:DeleteDataset
	databrew:DeleteJob
	databrew:DeleteProject
	databrew:DeleteRecipeVersion
	databrew:DeleteRuleset
	databrew:DeleteSchedule
	databrew:DescribeDataset
	databrew:DescribeJob
	databrew:DescribeJobRun
	databrew:DescribeProject
	databrew:DescribeRecipe
	databrew:DescribeRuleset
	databrew:DescribeSchedule

Service prefix	Actions
	databrew:ListDatasets
	databrew:ListJobRuns
	databrew:ListJobs
	databrew:ListProjects
	databrew:ListRecipeVersions
	databrew:ListRecipes
	databrew:ListRulesets
	databrew:ListSchedules
	databrew:PublishRecipe
	databrew:SendProjectSessionAction
	databrew:StartJobRun
	databrew:StartProjectSession
	databrew:StopJobRun
	databrew:UpdateDataset
	databrew:UpdateProfileJob
	databrew:UpdateProject
	databrew:UpdateRecipe
	databrew:UpdateRecipeJob
	databrew:UpdateRuleset
	databrew:UpdateSchedule

Service prefix	Actions
dataexchange	dataexchange:AcceptDataGrant dataexchange:CancelJob dataexchange:CreateDataGrant dataexchange:CreateDataSet dataexchange:CreateEventAction dataexchange:CreateJob dataexchange:CreateRevision dataexchange:DeleteAsset dataexchange:DeleteDataGrant dataexchange:DeleteEventAction dataexchange:DeleteRevision dataexchange:GetDataGrant dataexchange:GetEventAction dataexchange:GetJob dataexchange:GetReceivedDataGrant dataexchange:ListDataGrants dataexchange:ListDataSetRevisions dataexchange:ListDataSets dataexchange:ListEventActions dataexchange:ListJobs dataexchange:ListReceivedDataGrants

Service prefix	Actions
	dataexchange:ListRevisionAssets dataexchange:RevokeRevision dataexchange:SendDataSetNotification dataexchange:StartJob dataexchange:UpdateAsset dataexchange:UpdateDataSet dataexchange:UpdateEventAction dataexchange:UpdateRevision

Service prefix	Actions
datapipeline	datapipeline:ActivatePipeline
	datapipeline:CreatePipeline
	datapipeline:DeactivatePipeline
	datapipeline:DeletePipeline
	datapipeline:DescribeObjects
	datapipeline:DescribePipelines
	datapipeline:EvaluateExpression
	datapipeline:GetPipelineDefinition
	datapipeline:ListPipelines
	datapipeline:PollForTask
	datapipeline:PutPipelineDefinition
	datapipeline:QueryObjects
	datapipeline:ReportTaskProgress
	datapipeline:ReportTaskRunnerHeartbeat
	datapipeline:SetStatus
	datapipeline:SetTaskStatus
	datapipeline:ValidatePipelineDefinition

Service prefix	Actions
dax	dax:CreateCluster dax:DecreaseReplicationFactor dax:DeleteCluster dax:DeleteParameterGroup dax:DeleteSubnetGroup dax:DescribeClusters dax:DescribeDefaultParameters dax:DescribeEvents dax:DescribeParameterGroups dax:DescribeParameters dax:DescribeSubnetGroups dax:IncreaseReplicationFactor dax:RebootNode dax:UpdateCluster dax:UpdateParameterGroup dax:UpdateSubnetGroup

Service prefix	Actions
devicefarm	devicefarm:CreateDevicePool
	devicefarm:CreateInstanceProfile
	devicefarm:CreateNetworkProfile
	devicefarm:CreateProject
	devicefarm:CreateRemoteAccessSession
	devicefarm:CreateTestGridProject
	devicefarm:CreateTestGridUrl
	devicefarm:CreateUpload
	devicefarm:CreateVPCEConfiguration
	devicefarm:DeleteDevicePool
	devicefarm:DeleteInstanceProfile
	devicefarm:DeleteNetworkProfile
	devicefarm:DeleteProject
	devicefarm:DeleteRemoteAccessSession
	devicefarm:DeleteRun
	devicefarm:DeleteTestGridProject
	devicefarm:DeleteUpload
	devicefarm:DeleteVPCEConfiguration
	devicefarm:GetAccountSettings
	devicefarm:GetDevice
	devicefarm:GetDeviceInstance

Service prefix	Actions
	devicefarm:GetDevicePool
	devicefarm:GetDevicePoolCompatibility
	devicefarm:GetInstanceProfile
	devicefarm:GetJob
	devicefarm:GetNetworkProfile
	devicefarm:GetOfferingStatus
	devicefarm:GetProject
	devicefarm:GetRemoteAccessSession
	devicefarm:GetRun
	devicefarm:GetSuite
	devicefarm:GetTest
	devicefarm:GetTestGridProject
	devicefarm:GetTestGridSession
	devicefarm:GetUpload
	devicefarm:GetVPCEConfiguration
	devicefarm:ListArtifacts
	devicefarm:ListDeviceInstances
	devicefarm:ListDevicePools
	devicefarm:ListDevices
	devicefarm:ListInstanceProfiles
	devicefarm:ListJobs

Service prefix	Actions
	devicefarm:ListNetworkProfiles
	devicefarm:ListOfferingPromotions
	devicefarm:ListOfferingTransactions
	devicefarm:ListOfferings
	devicefarm:ListProjects
	devicefarm:ListRemoteAccessSessions
	devicefarm:ListRuns
	devicefarm:ListSamples
	devicefarm:ListSuites
	devicefarm:ListTestGridProjects
	devicefarm:ListTestGridSessionActions
	devicefarm:ListTestGridSessionArtifacts
	devicefarm:ListTestGridSessions
	devicefarm:ListTests
	devicefarm:ListUniqueProblems
	devicefarm:ListUploads
	devicefarm:ListVPCEConfigurations
	devicefarm:PurchaseOffering
	devicefarm:RenewOffering
	devicefarm:ScheduleRun
	devicefarm:StopJob

Service prefix	Actions
	devicefarm:StopRemoteAccessSession
	devicefarm:StopRun
	devicefarm:UpdateDeviceInstance
	devicefarm:UpdateDevicePool
	devicefarm:UpdateInstanceProfile
	devicefarm:UpdateNetworkProfile
	devicefarm:UpdateProject
	devicefarm:UpdateTestGridProject
	devicefarm:UpdateUpload
	devicefarm:UpdateVPCEConfiguration

Service prefix	Actions
devops-guru	devops-guru:AddNotificationChannel devops-guru:DeleteInsight devops-guru:DescribeAccountHealth devops-guru:DescribeAccountOverview devops-guru:DescribeAnomaly devops-guru:DescribeEventSourcesConfig devops-guru:DescribeFeedback devops-guru:DescribeInsight devops-guru:DescribeOrganizationHealth devops-guru:DescribeOrganizationOverview devops-guru:DescribeOrganizationResourceCollectionHealth devops-guru:DescribeResourceCollectionHealth devops-guru:DescribeServiceIntegration devops-guru:GetCostEstimation devops-guru:GetResourceCollection devops-guru:ListAnomaliesForInsight devops-guru:ListAnomalousLogGroups devops-guru:ListEvents devops-guru:ListInsights devops-guru:ListMonitoredResources devops-guru:ListNotificationChannels

Service prefix	Actions
	devops-guru:ListOrganizationInsights devops-guru:ListRecommendations devops-guru:PutFeedback devops-guru:RemoveNotificationChannel devops-guru:SearchInsights devops-guru:SearchOrganizationInsights devops-guru:StartCostEstimation devops-guru:UpdateEventSourcesConfig devops-guru:UpdateResourceCollection devops-guru:UpdateServiceIntegration

Service prefix	Actions
directconnect	directconnect:AcceptDirectConnectGatewayAssociationProposal
	directconnect:AllocateConnectionOnInterconnect
	directconnect:AllocateHostedConnection
	directconnect:AllocatePrivateVirtualInterface
	directconnect:AllocatePublicVirtualInterface
	directconnect:AllocateTransitVirtualInterface
	directconnect:AssociateConnectionWithLag
	directconnect:AssociateHostedConnection
	directconnect:AssociateMacSecKey
	directconnect:AssociateVirtualInterface
	directconnect:ConfirmConnection
	directconnect:ConfirmCustomerAgreement
	directconnect:ConfirmPrivateVirtualInterface
	directconnect:ConfirmPublicVirtualInterface
	directconnect:ConfirmTransitVirtualInterface
	directconnect:CreateBGPPeer
	directconnect:CreateConnection
	directconnect:CreateDirectConnectGateway
	directconnect:CreateDirectConnectGatewayAssociation
	directconnect:CreateDirectConnectGatewayAssociationProposal
	directconnect:CreateInterconnect

Service prefix	Actions
	directconnect:CreateLag
	directconnect:CreatePrivateVirtualInterface
	directconnect:CreatePublicVirtualInterface
	directconnect:CreateTransitVirtualInterface
	directconnect:DeleteBGPPeer
	directconnect:DeleteConnection
	directconnect:DeleteDirectConnectGateway
	directconnect:DeleteDirectConnectGatewayAssociation
	directconnect:DeleteDirectConnectGatewayAssociationProposal
	directconnect:DeleteInterconnect
	directconnect:DeleteLag
	directconnect:DeleteVirtualInterface
	directconnect:DescribeConnectionLoa
	directconnect:DescribeConnections
	directconnect:DescribeConnectionsOnInterconnect
	directconnect:DescribeCustomerMetadata
	directconnect:DescribeDirectConnectGatewayAssociationProposals
	directconnect:DescribeDirectConnectGatewayAssociations
	directconnect:DescribeDirectConnectGatewayAttachments
	directconnect:DescribeDirectConnectGateways
	directconnect:DescribeHostedConnections

Service prefix	Actions
	directconnect:DescribeInterconnectLoa
	directconnect:DescribeInterconnects
	directconnect:DescribeLags
	directconnect:DescribeLoa
	directconnect:DescribeLocations
	directconnect:DescribeRouterConfiguration
	directconnect:DescribeVirtualGateways
	directconnect:DescribeVirtualInterfaces
	directconnect:DisassociateConnectionFromLag
	directconnect:DisassociateMacSecKey
	directconnect:ListVirtualInterfaceTestHistory
	directconnect:StartBgpFailoverTest
	directconnect:StopBgpFailoverTest
	directconnect:UpdateConnection
	directconnect:UpdateDirectConnectGateway
	directconnect:UpdateDirectConnectGatewayAssociation
	directconnect:UpdateLag
	directconnect:UpdateVirtualInterfaceAttributes

Service prefix	Actions
dlm	dlm:CreateLifecyclePolicy dlm:DeleteLifecyclePolicy dlm:GetLifecyclePolicies dlm:GetLifecyclePolicy dlm:UpdateLifecyclePolicy

Service prefix	Actions
dms	dms:ApplyPendingMaintenanceAction
	dms:AssociateExtensionPack
	dms:BatchStartRecommendations
	dms:CancelMetadataModelCreation
	dms:CancelReplicationTaskAssessmentRun
	dms:CreateDataProvider
	dms:CreateEndpoint
	dms:CreateEventSubscription
	dms:CreateInstanceProfile
	dms:CreateMigrationProject
	dms:CreateReplicationConfig
	dms:CreateReplicationInstance
	dms:CreateReplicationSubnetGroup
	dms:CreateReplicationTask
	dms:DeleteCertificate
	dms:DeleteConnection
	dms:DeleteDataMigration
	dms:DeleteDataProvider
	dms:DeleteEndpoint
	dms:DeleteEventSubscription
	dms:DeleteFleetAdvisorCollector

Service prefix	Actions
	dms:DeleteFleetAdvisorDatabases
	dms:DeleteInstanceProfile
	dms:DeleteMigrationProject
	dms:DeleteReplicationConfig
	dms:DeleteReplicationInstance
	dms:DeleteReplicationSubnetGroup
	dms:DeleteReplicationTask
	dms:DeleteReplicationTaskAssessmentRun
	dms:DescribeAccountAttributes
	dms:DescribeApplicableIndividualAssessments
	dms:DescribeCertificates
	dms:DescribeConnections
	dms:DescribeDataMigrations
	dms:DescribeEndpointSettings
	dms:DescribeEndpointTypes
	dms:DescribeEndpoints
	dms:DescribeEngineVersions
	dms:DescribeEventCategories
	dms:DescribeEventSubscriptions
	dms:DescribeEvents
	dms:DescribeFleetAdvisorCollectors

Service prefix	Actions
	dms:DescribeFleetAdvisorDatabases
	dms:DescribeFleetAdvisorLsaAnalysis
	dms:DescribeFleetAdvisorSchemaObjectSummary
	dms:DescribeFleetAdvisorSchemas
	dms:DescribeMetadataModel
	dms:DescribeMetadataModelChildren
	dms:DescribeMetadataModelCreations
	dms:DescribeMetadataModelImports
	dms:DescribeOrderableReplicationInstances
	dms:DescribePendingMaintenanceActions
	dms:DescribeRecommendationLimitations
	dms:DescribeRecommendations
	dms:DescribeRefreshSchemasStatus
	dms:DescribeReplicationConfigs
	dms:DescribeReplicationInstanceTaskLogs
	dms:DescribeReplicationInstances
	dms:DescribeReplicationSubnetGroups
	dms:DescribeReplicationTableStatistics
	dms:DescribeReplicationTaskAssessmentResults
	dms:DescribeReplicationTaskAssessmentRuns
	dms:DescribeReplicationTaskIndividualAssessments

Service prefix	Actions
	dms:DescribeReplicationTasks
	dms:DescribeReplications
	dms:DescribeSchemas
	dms:DescribeTableStatistics
	dms:ExportMetadataModelAssessment
	dms:ImportCertificate
	dms:ListDataProviders
	dms:ListExtensionPacks
	dms:ListInstanceProfiles
	dms:ListMetadataModelAssessments
	dms:ListMetadataModelConversions
	dms:ListMetadataModelExports
	dms:ListMigrationProjects
	dms:ModifyDataMigration
	dms:ModifyEndpoint
	dms:ModifyEventSubscription
	dms:ModifyReplicationConfig
	dms:ModifyReplicationInstance
	dms:ModifyReplicationSubnetGroup
	dms:ModifyReplicationTask
	dms:MoveReplicationTask

Service prefix	Actions
	dms:RebootReplicationInstance
	dms:RefreshSchemas
	dms:ReloadReplicationTables
	dms:ReloadTables
	dms:RunFleetAdvisorLsaAnalysis
	dms:StartMetadataModelAssessment
	dms:StartMetadataModelConversion
	dms:StartMetadataModelCreation
	dms:StartMetadataModelExportAsScripts
	dms:StartMetadataModelExportToTarget
	dms:StartRecommendations
	dms:StartReplication
	dms:StartReplicationTask
	dms:StartReplicationTaskAssessment
	dms:StopDataMigration
	dms:StopReplicationTask
	dms:TestConnection
	dms:UpdateConversionConfiguration
	dms:UpdateDataProvider
	dms:UpdateInstanceProfile
	dms:UpdateMigrationProject

Service prefix	Actions
	dms:UpdateSubscriptionsToEventBridge
docdb-elastic	docdb-elastic:ApplyPendingMaintenanceAction docdb-elastic:CopyClusterSnapshot docdb-elastic:DeleteCluster docdb-elastic:DeleteClusterSnapshot docdb-elastic:GetCluster docdb-elastic:GetClusterSnapshot docdb-elastic:GetPendingMaintenanceAction docdb-elastic:ListClusterSnapshots docdb-elastic:ListClusters docdb-elastic:ListPendingMaintenanceActions docdb-elastic:RestoreClusterFromSnapshot docdb-elastic:StartCluster docdb-elastic:StopCluster docdb-elastic:UpdateCluster

Service prefix	Actions
dynamodb	dynamodb:CreateBackup
	dynamodb:CreateGlobalTable
	dynamodb:CreateTable
	dynamodb:DeleteBackup
	dynamodb:DeleteTable
	dynamodb:DescribeBackup
	dynamodb:DescribeContinuousBackups
	dynamodb:DescribeContributorInsights
	dynamodb:DescribeEndpoints
	dynamodb:DescribeExport
	dynamodb:DescribeGlobalTable
	dynamodb:DescribeGlobalTableSettings
	dynamodb:DescribeImport
	dynamodb:DescribeKinesisStreamingDestination
	dynamodb:DescribeLimits
	dynamodb:DescribeStream
	dynamodb:DescribeTable
	dynamodb:DescribeTableReplicaAutoScaling
	dynamodb:DescribeTimeToLive
	dynamodb:DisableKinesisStreamingDestination
	dynamodb:EnableKinesisStreamingDestination

Service prefix	Actions
	dynamodb:ExportTableToPointInTime
	dynamodb:GetResourcePolicy
	dynamodb:ImportTable
	dynamodb:ListBackups
	dynamodb:ListContributorInsights
	dynamodb:ListExports
	dynamodb:ListGlobalTables
	dynamodb:ListImports
	dynamodb:ListStreams
	dynamodb:ListTables
	dynamodb:RestoreTableFromBackup
	dynamodb:RestoreTableToPointInTime
	dynamodb:UpdateContinuousBackups
	dynamodb:UpdateContributorInsights
	dynamodb:UpdateGlobalTable
	dynamodb:UpdateGlobalTableSettings
	dynamodb:UpdateKinesisStreamingDestination
	dynamodb:UpdateTable
	dynamodb:UpdateTableReplicaAutoScaling
	dynamodb:UpdateTimeToLive

Service prefix	Actions
ebs	ebs:CompleteSnapshot ebs:StartSnapshot

Service prefix	Actions
ec2	ec2:AcceptAddressTransfer
	ec2:AcceptCapacityReservationBillingOwnership
	ec2:AcceptReservedInstancesExchangeQuote
	ec2:AcceptTransitGatewayClientVpnAttachment
	ec2:AcceptTransitGatewayMulticastDomainAssociations
	ec2:AcceptTransitGatewayPeeringAttachment
	ec2:AcceptTransitGatewayVpcAttachment
	ec2:AcceptVpcEndpointConnections
	ec2:AcceptVpcPeeringConnection
	ec2:AdvertiseByoipCidr
	ec2:AllocateAddress
	ec2:AllocateHosts
	ec2:AllocateIpamPoolCidr
	ec2:ApplySecurityGroupsToClientVpnTargetNetwork
	ec2:AssignIpv6Addresses
	ec2:AssignPrivateIpAddresses
	ec2:AssignPrivateNatGatewayAddress
	ec2:AssociateAddress
	ec2:AssociateCapacityReservationBillingOwner
	ec2:AssociateClientVpnTargetNetwork
	ec2:AssociateDhcpOptions

Service prefix	Actions
	ec2:AssociateEnclaveCertificateIamRole
	ec2:AssociateIamInstanceProfile
	ec2:AssociateInstanceEventWindow
	ec2:AssociateIpamByoasn
	ec2:AssociateIpamResourceDiscovery
	ec2:AssociateNatGatewayAddress
	ec2:AssociateRouteServer
	ec2:AssociateRouteTable
	ec2:AssociateSecurityGroupVpc
	ec2:AssociateSubnetCidrBlock
	ec2:AssociateTransitGatewayMulticastDomain
	ec2:AssociateTransitGatewayPolicyTable
	ec2:AssociateTransitGatewayRouteTable
	ec2:AssociateTrunkInterface
	ec2:AssociateVpcCidrBlock
	ec2:AttachClassicLinkVpc
	ec2:AttachImageWatermark
	ec2:AttachInternetGateway
	ec2:AttachNetworkInterface
	ec2:AttachVerifiedAccessTrustProvider
	ec2:AttachVolume

Service prefix	Actions
	ec2:AttachVpnGateway
	ec2:AuthorizeClientVpnIngress
	ec2:AuthorizeSecurityGroupEgress
	ec2:AuthorizeSecurityGroupIngress
	ec2:BundleInstance
	ec2:CancelBundleTask
	ec2:CancelCapacityReservation
	ec2:CancelCapacityReservationFleets
	ec2:CancelConversionTask
	ec2:CancelDeclarativePoliciesReport
	ec2:CancelExportTask
	ec2:CancelImageLaunchPermission
	ec2:CancelImportTask
	ec2:CancelReservedInstancesListing
	ec2:CancelSpotFleetRequests
	ec2:CancelSpotInstanceRequests
	ec2:ConfirmProductInstance
	ec2:CopyFpgaImage
	ec2:CopyImage
	ec2:CopySnapshot
	ec2:CopyVolumes

Service prefix	Actions
	ec2:CreateCapacityManagerDataExport
	ec2:CreateCapacityReservation
	ec2:CreateCapacityReservationBySplitting
	ec2:CreateCapacityReservationCancellationQuote
	ec2:CreateCapacityReservationFleet
	ec2:CreateCarrierGateway
	ec2:CreateClientVpnEndpoint
	ec2:CreateClientVpnRoute
	ec2:CreateCoipCidr
	ec2:CreateCoipPool
	ec2:CreateCustomerGateway
	ec2:CreateDefaultSubnet
	ec2:CreateDefaultVpc
	ec2:CreateDelegateMacVolumeOwnershipTask
	ec2:CreateDhcpOptions
	ec2:CreateEgressOnlyInternetGateway
	ec2:CreateFleet
	ec2:CreateFlowLogs
	ec2:CreateFpgaImage
	ec2:CreateImage
	ec2:CreateImageUsageReport

Service prefix	Actions
	ec2:CreateInstanceConnectEndpoint
	ec2:CreateInstanceEventWindow
	ec2:CreateInstanceExportTask
	ec2:CreateInternetGateway
	ec2:CreateInterruptibleCapacityReservationAllocation
	ec2:CreateIpam
	ec2:CreateIpamExternalResourceVerificationToken
	ec2:CreateIpamPolicy
	ec2:CreateIpamPool
	ec2:CreateIpamPrefixListResolver
	ec2:CreateIpamPrefixListResolverTarget
	ec2:CreateIpamResourceDiscovery
	ec2:CreateIpamScope
	ec2:CreateKeyPair
	ec2:CreateLaunchTemplateVersion
	ec2:CreateLocalGatewayRoute
	ec2:CreateLocalGatewayRouteTable
	ec2:CreateLocalGatewayRouteTableVirtualInterfaceGroupAssociation
	ec2:CreateLocalGatewayRouteTableVpcAssociation
	ec2:CreateLocalGatewayVirtualInterface

Service prefix	Actions
	ec2:CreateLocalGatewayVirtualInterfaceGroup
	ec2:CreateMacSystemIntegrityProtectionModificationTask
	ec2:CreateManagedPrefixList
	ec2:CreateNatGateway
	ec2:CreateNetworkAcl
	ec2:CreateNetworkAclEntry
	ec2:CreateNetworkInsightsAccessScope
	ec2:CreateNetworkInsightsPath
	ec2:CreateNetworkInterface
	ec2:CreateNetworkInterfacePermission
	ec2:CreatePlacementGroup
	ec2:CreatePublicIpv4Pool
	ec2:CreateReplaceRootVolumeTask
	ec2:CreateReservedInstancesListing
	ec2:CreateRestoreImageTask
	ec2:CreateRoute
	ec2:CreateRouteServer
	ec2:CreateRouteServerEndpoint
	ec2:CreateRouteServerPeer
	ec2:CreateRouteTable
	ec2:CreateSecondaryNetwork

Service prefix	Actions
	ec2:CreateSecondarySubnet
	ec2:CreateSecurityGroup
	ec2:CreateSnapshots
	ec2:CreateSpotDatafeedSubscription
	ec2:CreateStoreImageTask
	ec2:CreateSubnet
	ec2:CreateSubnetCidrReservation
	ec2:CreateTrafficMirrorFilter
	ec2:CreateTrafficMirrorFilterRule
	ec2:CreateTrafficMirrorSession
	ec2:CreateTrafficMirrorTarget
	ec2:CreateTransitGateway
	ec2:CreateTransitGatewayConnect
	ec2:CreateTransitGatewayConnectPeer
	ec2:CreateTransitGatewayMeteringPolicy
	ec2:CreateTransitGatewayMeteringPolicyEntry
	ec2:CreateTransitGatewayMulticastDomain
	ec2:CreateTransitGatewayPeeringAttachment
	ec2:CreateTransitGatewayPolicyTable
	ec2:CreateTransitGatewayPrefixListReference
	ec2:CreateTransitGatewayRoute

Service prefix	Actions
	ec2:CreateTransitGatewayRouteTable
	ec2:CreateTransitGatewayRouteTableAnnouncement
	ec2:CreateTransitGatewayVpcAttachment
	ec2:CreateVerifiedAccessEndpoint
	ec2:CreateVerifiedAccessGroup
	ec2:CreateVerifiedAccessInstance
	ec2:CreateVerifiedAccessTrustProvider
	ec2:CreateVolume
	ec2:CreateVpc
	ec2:CreateVpcBlockPublicAccessExclusion
	ec2:CreateVpcEncryptionControl
	ec2:CreateVpcEndpoint
	ec2:CreateVpcEndpointConnectionNotification
	ec2:CreateVpcEndpointServiceConfiguration
	ec2:CreateVpcPeeringConnection
	ec2:CreateVpnConcentrator
	ec2:CreateVpnConnection
	ec2:CreateVpnConnectionRoute
	ec2:CreateVpnGateway
	ec2:DeleteCapacityManagerDataExport
	ec2:DeleteCarrierGateway

Service prefix	Actions
	ec2:DeleteClientVpnEndpoint
	ec2:DeleteClientVpnRoute
	ec2:DeleteCoipCidr
	ec2:DeleteCoipPool
	ec2:DeleteCustomerGateway
	ec2:DeleteDhcpOptions
	ec2:DeleteEgressOnlyInternetGateway
	ec2:DeleteFleets
	ec2:DeleteFlowLogs
	ec2:DeleteFpgaImage
	ec2:DeleteImageUsageReport
	ec2:DeleteInstanceConnectEndpoint
	ec2:DeleteInstanceEventWindow
	ec2:DeleteInternetGateway
	ec2:DeleteIpam
	ec2:DeleteIpamExternalResourceVerificationToken
	ec2:DeleteIpamPolicy
	ec2:DeleteIpamPool
	ec2:DeleteIpamPrefixListResolver
	ec2:DeleteIpamPrefixListResolverTarget
	ec2:DeleteIpamResourceDiscovery

Service prefix	Actions
	ec2:DeleteIpamScope
	ec2:DeleteKeyPair
	ec2:DeleteLaunchTemplate
	ec2:DeleteLaunchTemplateVersions
	ec2:DeleteLocalGatewayRoute
	ec2:DeleteLocalGatewayRouteTable
	ec2:DeleteLocalGatewayRouteTableVirtualInterfaceGroupAssociation
	ec2:DeleteLocalGatewayRouteTableVpcAssociation
	ec2:DeleteLocalGatewayVirtualInterface
	ec2:DeleteLocalGatewayVirtualInterfaceGroup
	ec2:DeleteManagedPrefixList
	ec2:DeleteNatGateway
	ec2:DeleteNetworkAcl
	ec2:DeleteNetworkAclEntry
	ec2:DeleteNetworkInsightsAccessScope
	ec2:DeleteNetworkInsightsAccessScopeAnalysis
	ec2:DeleteNetworkInsightsAnalysis
	ec2:DeleteNetworkInsightsPath
	ec2:DeleteNetworkInterface
	ec2:DeleteNetworkInterfacePermission

Service prefix	Actions
	ec2:DeletePlacementGroup
	ec2:DeletePublicIpv4Pool
	ec2:DeleteQueuedReservedInstances
	ec2:DeleteRoute
	ec2:DeleteRouteServer
	ec2:DeleteRouteServerEndpoint
	ec2:DeleteRouteServerPeer
	ec2:DeleteRouteTable
	ec2:DeleteSecondaryNetwork
	ec2:DeleteSecondarySubnet
	ec2:DeleteSecurityGroup
	ec2:DeleteSpotDatafeedSubscription
	ec2:DeleteSubnet
	ec2:DeleteSubnetCidrReservation
	ec2:DeleteTrafficMirrorFilter
	ec2:DeleteTrafficMirrorFilterRule
	ec2:DeleteTrafficMirrorSession
	ec2:DeleteTrafficMirrorTarget
	ec2:DeleteTransitGateway
	ec2:DeleteTransitGatewayClientVpnAttachment
	ec2:DeleteTransitGatewayConnect

Service prefix	Actions
	ec2:DeleteTransitGatewayConnectPeer
	ec2:DeleteTransitGatewayMeteringPolicy
	ec2:DeleteTransitGatewayMeteringPolicyEntry
	ec2:DeleteTransitGatewayMulticastDomain
	ec2:DeleteTransitGatewayPeeringAttachment
	ec2:DeleteTransitGatewayPolicyTable
	ec2:DeleteTransitGatewayPrefixListReference
	ec2:DeleteTransitGatewayRoute
	ec2:DeleteTransitGatewayRouteTable
	ec2:DeleteTransitGatewayRouteTableAnnouncement
	ec2:DeleteTransitGatewayVpcAttachment
	ec2:DeleteVerifiedAccessEndpoint
	ec2:DeleteVerifiedAccessGroup
	ec2:DeleteVerifiedAccessInstance
	ec2:DeleteVerifiedAccessTrustProvider
	ec2:DeleteVolume
	ec2:DeleteVpc
	ec2:DeleteVpcBlockPublicAccessExclusion
	ec2:DeleteVpcEncryptionControl
	ec2:DeleteVpcEndpointConnectionNotifications
	ec2:DeleteVpcEndpointServiceConfigurations

Service prefix	Actions
	ec2:DeleteVpcEndpoints
	ec2:DeleteVpcPeeringConnection
	ec2:DeleteVpnConcentrator
	ec2:DeleteVpnConnection
	ec2:DeleteVpnConnectionRoute
	ec2:DeleteVpnGateway
	ec2:DeprovisionByoipCidr
	ec2:DeprovisionIpamByoasn
	ec2:DeprovisionIpamPoolCidr
	ec2:DeprovisionPublicIpv4PoolCidr
	ec2:DeregisterImage
	ec2:DeregisterInstanceEventNotificationAttributes
	ec2:DeregisterTransitGatewayMulticastGroupMembers
	ec2:DeregisterTransitGatewayMulticastGroupSources
	ec2:DescribeAccountAttributes
	ec2:DescribeAddressTransfers
	ec2:DescribeAddresses
	ec2:DescribeAddressesAttribute
	ec2:DescribeAggregateIdFormat
	ec2:DescribeAvailabilityZones
	ec2:DescribeAwsNetworkPerformanceMetricSubscriptions

Service prefix	Actions
	ec2:DescribeBundleTasks
	ec2:DescribeByoipCidrs
	ec2:DescribeCapacityBlockExtensionHistory
	ec2:DescribeCapacityBlockExtensionOfferings
	ec2:DescribeCapacityBlockStatus
	ec2:DescribeCapacityBlocks
	ec2:DescribeCapacityManagerDataExports
	ec2:DescribeCapacityReservationBillingRequests
	ec2:DescribeCapacityReservationCancellationQuotes
	ec2:DescribeCapacityReservationFleets
	ec2:DescribeCapacityReservationTopology
	ec2:DescribeCapacityReservations
	ec2:DescribeCarrierGateways
	ec2:DescribeClassicLinkInstances
	ec2:DescribeClientVpnAuthorizationRules
	ec2:DescribeClientVpnConnections
	ec2:DescribeClientVpnEndpoints
	ec2:DescribeClientVpnRoutes
	ec2:DescribeClientVpnTargetNetworks
	ec2:DescribeCoipPools
	ec2:DescribeConversionTasks

Service prefix	Actions
	ec2:DescribeCustomerGateways
	ec2:DescribeDeclarativePoliciesReports
	ec2:DescribeDhcpOptions
	ec2:DescribeEgressOnlyInternetGateways
	ec2:DescribeElasticGpus
	ec2:DescribeExportImageTasks
	ec2:DescribeExportTasks
	ec2:DescribeFastLaunchImages
	ec2:DescribeFastSnapshotRestores
	ec2:DescribeFleetHistory
	ec2:DescribeFleetInstances
	ec2:DescribeFleets
	ec2:DescribeFlowLogs
	ec2:DescribeFpgaImageAttribute
	ec2:DescribeFpgaImages
	ec2:DescribeHostReservationOfferings
	ec2:DescribeHostReservations
	ec2:DescribeHosts
	ec2:DescribeIamInstanceProfileAssociations
	ec2:DescribeIdFormat
	ec2:DescribeIdentityIdFormat

Service prefix	Actions
	ec2:DescribeImageAttribute
	ec2:DescribeImageReferences
	ec2:DescribeImageUsageReportEntries
	ec2:DescribeImageUsageReports
	ec2:DescribeImportImageTasks
	ec2:DescribeImportSnapshotTasks
	ec2:DescribeInstanceConnectEndpoints
	ec2:DescribeInstanceCreditSpecifications
	ec2:DescribeInstanceEventNotificationAttributes
	ec2:DescribeInstanceEventWindows
	ec2:DescribeInstanceImageMetadata
	ec2:DescribeInstanceSqlHaHistoryStates
	ec2:DescribeInstanceSqlHaStates
	ec2:DescribeInstanceTopology
	ec2:DescribeInstanceTypes
	ec2:DescribeInternetGateways
	ec2:DescribeIpamByoasn
	ec2:DescribeIpamExternalResourceVerificationTokens
	ec2:DescribeIpamPolicies
	ec2:DescribeIpamPoolAllocations
	ec2:DescribeIpamPools

Service prefix	Actions
	ec2:DescribeIpamPrefixListResolverTargets
	ec2:DescribeIpamPrefixListResolvers
	ec2:DescribeIpamResourceDiscoveries
	ec2:DescribeIpamResourceDiscoveryAssociations
	ec2:DescribeIpamScopes
	ec2:DescribeIpams
	ec2:DescribeIpv6Pools
	ec2:DescribeKeyPairs
	ec2:DescribeLocalGatewayRouteTableVirtualInterfaceGroupAssociations
	ec2:DescribeLocalGatewayRouteTableVpcAssociations
	ec2:DescribeLocalGatewayRouteTables
	ec2:DescribeLocalGatewayVirtualInterfaceGroups
	ec2:DescribeLocalGatewayVirtualInterfaces
	ec2:DescribeLocalGateways
	ec2:DescribeLockedSnapshots
	ec2:DescribeMacHosts
	ec2:DescribeMacModificationTasks
	ec2:DescribeManagedPrefixLists
	ec2:DescribeMovingAddresses
	ec2:DescribeNatGateways

Service prefix	Actions
	ec2:DescribeNetworkAcls
	ec2:DescribeNetworkInsightsAccessScopeAnalyses
	ec2:DescribeNetworkInsightsAccessScopes
	ec2:DescribeNetworkInsightsAnalyses
	ec2:DescribeNetworkInsightsPaths
	ec2:DescribeNetworkInterfaceAttribute
	ec2:DescribeNetworkInterfacePermissions
	ec2:DescribeNetworkInterfaces
	ec2:DescribeOutpostLags
	ec2:DescribePlacementGroups
	ec2:DescribePrefixLists
	ec2:DescribePrincipalIdFormat
	ec2:DescribePublicIpv4Pools
	ec2:DescribeRegions
	ec2:DescribeReplaceRootVolumeTasks
	ec2:DescribeReservedInstances
	ec2:DescribeReservedInstancesListings
	ec2:DescribeReservedInstancesModifications
	ec2:DescribeReservedInstancesOfferings
	ec2:DescribeRouteServerEndpoints
	ec2:DescribeRouteServerPeers

Service prefix	Actions
	ec2:DescribeRouteServers
	ec2:DescribeRouteTables
	ec2:DescribeScheduledInstanceAvailability
	ec2:DescribeScheduledInstances
	ec2:DescribeSecondaryInterfaces
	ec2:DescribeSecondaryNetworks
	ec2:DescribeSecondarySubnets
	ec2:DescribeSecurityGroupReferences
	ec2:DescribeSecurityGroupRules
	ec2:DescribeSecurityGroupVpcAssociations
	ec2:DescribeSecurityGroups
	ec2:DescribeServiceLinkVirtualInterfaces
	ec2:DescribeSnapshotAttribute
	ec2:DescribeSnapshotTierStatus
	ec2:DescribeSpotDatafeedSubscription
	ec2:DescribeSpotFleetInstances
	ec2:DescribeSpotFleetRequestHistory
	ec2:DescribeSpotFleetRequests
	ec2:DescribeSpotInstanceRequests
	ec2:DescribeSpotPriceHistory
	ec2:DescribeStaleSecurityGroups

Service prefix	Actions
	ec2:DescribeStoreImageTasks
	ec2:DescribeTrafficMirrorFilterRules
	ec2:DescribeTrafficMirrorFilters
	ec2:DescribeTrafficMirrorSessions
	ec2:DescribeTrafficMirrorTargets
	ec2:DescribeTransitGatewayAttachments
	ec2:DescribeTransitGatewayConnectPeers
	ec2:DescribeTransitGatewayConnects
	ec2:DescribeTransitGatewayMeteringPolicies
	ec2:DescribeTransitGatewayMulticastDomains
	ec2:DescribeTransitGatewayPeeringAttachments
	ec2:DescribeTransitGatewayPolicyTables
	ec2:DescribeTransitGatewayRouteTableAnnouncements
	ec2:DescribeTransitGatewayRouteTables
	ec2:DescribeTransitGatewayVpcAttachments
	ec2:DescribeTransitGateways
	ec2:DescribeTrunkInterfaceAssociations
	ec2:DescribeVerifiedAccessEndpoints
	ec2:DescribeVerifiedAccessGroups
	ec2:DescribeVerifiedAccessInstanceLoggingConfigurations
	ec2:DescribeVerifiedAccessInstances

Service prefix	Actions
	ec2:DescribeVerifiedAccessTrustProviders
	ec2:DescribeVolumeAttribute
	ec2:DescribeVolumeStatus
	ec2:DescribeVolumes
	ec2:DescribeVolumesModifications
	ec2:DescribeVpcAttribute
	ec2:DescribeVpcBlockPublicAccessExclusions
	ec2:DescribeVpcBlockPublicAccessOptions
	ec2:DescribeVpcClassicLink
	ec2:DescribeVpcClassicLinkDnsSupport
	ec2:DescribeVpcEncryptionControls
	ec2:DescribeVpcEndpointAssociations
	ec2:DescribeVpcEndpointConnectionNotifications
	ec2:DescribeVpcEndpointConnections
	ec2:DescribeVpcEndpointServiceConfigurations
	ec2:DescribeVpcEndpointServicePermissions
	ec2:DescribeVpcEndpointServices
	ec2:DescribeVpcEndpoints
	ec2:DescribeVpcPeeringConnections
	ec2:DescribeVpcs
	ec2:DescribeVpnConcentrators

Service prefix	Actions
	ec2:DescribeVpnConnections
	ec2:DescribeVpnGateways
	ec2:DetachClassicLinkVpc
	ec2:DetachImageWatermark
	ec2:DetachInternetGateway
	ec2:DetachNetworkInterface
	ec2:DetachVerifiedAccessTrustProvider
	ec2:DetachVolume
	ec2:DetachVpnGateway
	ec2:DisableAddressTransfer
	ec2:DisableAllowedImagesSettings
	ec2:DisableAwsNetworkPerformanceMetricSubscription
	ec2:DisableCapacityManager
	ec2:DisableEbsEncryptionByDefault
	ec2:DisableFastLaunch
	ec2:DisableFastSnapshotRestores
	ec2:DisableImage
	ec2:DisableImageBlockPublicAccess
	ec2:DisableImageDeprecation
	ec2:DisableImageDeregistrationProtection
	ec2:DisableInstanceSqlHaStandbyDetections

Service prefix	Actions
	ec2:DisableIpamOrganizationAdminAccount
	ec2:DisableIpamPolicy
	ec2:DisableRouteServerPropagation
	ec2:DisableSerialConsoleAccess
	ec2:DisableSnapshotBlockPublicAccess
	ec2:DisableTransitGatewayRouteTablePropagation
	ec2:DisableVgwRoutePropagation
	ec2:DisableVpcClassicLink
	ec2:DisableVpcClassicLinkDnsSupport
	ec2:DisassociateAddress
	ec2:DisassociateCapacityReservationBillingOwner
	ec2:DisassociateClientVpnTargetNetwork
	ec2:DisassociateEnclaveCertificateIamRole
	ec2:DisassociateIamInstanceProfile
	ec2:DisassociateInstanceEventWindow
	ec2:DisassociateIpamByoasn
	ec2:DisassociateIpamResourceDiscovery
	ec2:DisassociateNatGatewayAddress
	ec2:DisassociateRouteServer
	ec2:DisassociateRouteTable
	ec2:DisassociateSecurityGroupVpc

Service prefix	Actions
	ec2:DisassociateSubnetCidrBlock
	ec2:DisassociateTransitGatewayMulticastDomain
	ec2:DisassociateTransitGatewayPolicyTable
	ec2:DisassociateTransitGatewayRouteTable
	ec2:DisassociateTrunkInterface
	ec2:DisassociateVpcCidrBlock
	ec2:EnableAddressTransfer
	ec2:EnableAllowedImagesSettings
	ec2:EnableAwsNetworkPerformanceMetricSubscription
	ec2:EnableCapacityManager
	ec2:EnableEbsEncryptionByDefault
	ec2:EnableFastLaunch
	ec2:EnableFastSnapshotRestores
	ec2:EnableImage
	ec2:EnableImageBlockPublicAccess
	ec2:EnableImageDeprecation
	ec2:EnableImageDeregistrationProtection
	ec2:EnableInstanceSqlHaStandbyDetections
	ec2:EnableIpamOrganizationAdminAccount
	ec2:EnableIpamPolicy
	ec2:EnableReachabilityAnalyzerOrganizationSharing

Service prefix	Actions
	ec2:EnableRouteServerPropagation
	ec2:EnableSerialConsoleAccess
	ec2:EnableSnapshotBlockPublicAccess
	ec2:EnableTransitGatewayRouteTablePropagation
	ec2:EnableVgwRoutePropagation
	ec2:EnableVolumeIO
	ec2:EnableVpcClassicLink
	ec2:EnableVpcClassicLinkDnsSupport
	ec2:ExportClientVpnClientCertificateRevocationList
	ec2:ExportClientVpnClientConfiguration
	ec2:ExportImage
	ec2:ExportTransitGatewayRoutes
	ec2:ExportVerifiedAccessInstanceClientConfiguration
	ec2:GetActiveVpnTunnelStatus
	ec2:GetAllowedImagesSettings
	ec2:GetAssociatedEnclaveCertificateIamRoles
	ec2:GetAssociatedIpv6PoolCidrs
	ec2:GetAwsNetworkPerformanceData
	ec2:GetCapacityManagerAttributes
	ec2:GetCapacityManagerMetricData
	ec2:GetCapacityManagerMetricDimensions

Service prefix	Actions
	ec2:GetCapacityReservationUsage
	ec2:GetCoipPoolUsage
	ec2:GetConsoleOutput
	ec2:GetConsoleScreenshot
	ec2:GetDeclarativePoliciesReportSummary
	ec2:GetDefaultCreditSpecification
	ec2:GetEbsDefaultKmsKeyId
	ec2:GetEbsEncryptionByDefault
	ec2:GetEnabledIpamPolicy
	ec2:GetFlowLogsIntegrationTemplate
	ec2:GetGroupsForCapacityReservation
	ec2:GetHostReservationPurchasePreview
	ec2:GetImageAncestry
	ec2:GetImageBlockPublicAccessState
	ec2:GetInstanceMetadataDefaults
	ec2:GetInstanceTpmEkPub
	ec2:GetInstanceTypesFromInstanceRequirements
	ec2:GetInstanceUefiData
	ec2:GetIpamAddressHistory
	ec2:GetIpamDiscoveredAccounts
	ec2:GetIpamDiscoveredPublicAddresses

Service prefix	Actions
	ec2:GetIpamDiscoveredResourceCidrs
	ec2:GetIpamPolicyAllocationRules
	ec2:GetIpamPolicyOrganizationTargets
	ec2:GetIpamPoolAllocations
	ec2:GetIpamPoolCidrs
	ec2:GetIpamPrefixListResolverRules
	ec2:GetIpamPrefixListResolverVersionEntries
	ec2:GetIpamPrefixListResolverVersions
	ec2:GetIpamResourceCidrs
	ec2:GetLaunchTemplateData
	ec2:GetManagedPrefixListAssociations
	ec2:GetManagedPrefixListEntries
	ec2:GetManagedResourceVisibility
	ec2:GetNetworkInsightsAccessScopeAnalysisFindings
	ec2:GetNetworkInsightsAccessScopeContent
	ec2:GetPasswordData
	ec2:GetReservedInstancesExchangeQuote
	ec2:GetRouteServerAssociations
	ec2:GetRouteServerPropagations
	ec2:GetRouteServerRoutingDatabase
	ec2:GetSecurityGroupsForVpc

Service prefix	Actions
	ec2:GetSerialConsoleAccessStatus
	ec2:GetSnapshotBlockPublicAccessState
	ec2:GetSpotPlacementScores
	ec2:GetSubnetCidrReservations
	ec2:GetTransitGatewayAttachmentPropagations
	ec2:GetTransitGatewayMeteringPolicyEntries
	ec2:GetTransitGatewayMulticastDomainAssociations
	ec2:GetTransitGatewayPolicyTableAssociations
	ec2:GetTransitGatewayPolicyTableEntries
	ec2:GetTransitGatewayPrefixListReferences
	ec2:GetTransitGatewayRouteTableAssociations
	ec2:GetTransitGatewayRouteTablePropagations
	ec2:GetVerifiedAccessEndpointPolicy
	ec2:GetVerifiedAccessEndpointTargets
	ec2:GetVerifiedAccessGroupPolicy
	ec2:GetVpcResourcesBlockingEncryptionEnforcement
	ec2:GetVpnConnectionDeviceSampleConfiguration
	ec2:GetVpnConnectionDeviceTypes
	ec2:GetVpnTunnelReplacementStatus
	ec2:ImportClientVpnClientCertificateRevocationList
	ec2:ImportImage

Service prefix	Actions
	ec2:ImportInstance
	ec2:ImportKeyPair
	ec2:ImportSnapshot
	ec2:ImportVolume
	ec2:ListImagesInRecycleBin
	ec2:ListSnapshotsInRecycleBin
	ec2:ListVolumesInRecycleBin
	ec2:LockSnapshot
	ec2:ModifyAddressAttribute
	ec2:ModifyAvailabilityZoneGroup
	ec2:ModifyCapacityReservation
	ec2:ModifyCapacityReservationFleet
	ec2:ModifyClientVpnEndpoint
	ec2:ModifyDefaultCreditSpecification
	ec2:ModifyEbsDefaultKmsKeyId
	ec2:ModifyFleet
	ec2:ModifyFpgaImageAttribute
	ec2:ModifyHosts
	ec2:ModifyIdFormat
	ec2:ModifyIdentityIdFormat
	ec2:ModifyImageAttribute

Service prefix	Actions
	ec2:ModifyInstanceAttribute
	ec2:ModifyInstanceCapacityReservationAttributes
	ec2:ModifyInstanceConnectEndpoint
	ec2:ModifyInstanceCpuOptions
	ec2:ModifyInstanceCreditSpecification
	ec2:ModifyInstanceEventStartTime
	ec2:ModifyInstanceEventWindow
	ec2:ModifyInstanceMaintenanceOptions
	ec2:ModifyInstanceMetadataDefaults
	ec2:ModifyInstanceMetadataOptions
	ec2:ModifyInstanceNetworkPerformanceOptions
	ec2:ModifyInstancePlacement
	ec2:ModifyIpam
	ec2:ModifyIpamPolicyAllocationRules
	ec2:ModifyIpamPool
	ec2:ModifyIpamPoolAllocation
	ec2:ModifyIpamPrefixListResolver
	ec2:ModifyIpamPrefixListResolverTarget
	ec2:ModifyIpamResourceCidr
	ec2:ModifyIpamResourceDiscovery
	ec2:ModifyIpamScope

Service prefix	Actions
	ec2:ModifyLaunchTemplate
	ec2:ModifyLocalGatewayRoute
	ec2:ModifyManagedPrefixList
	ec2:ModifyManagedResourceVisibility
	ec2:ModifyNetworkInterfaceAttribute
	ec2:ModifyPrivateDnsNameOptions
	ec2:ModifyPublicIpDnsNameOptions
	ec2:ModifyReservedInstances
	ec2:ModifyRouteServer
	ec2:ModifySecurityGroupRules
	ec2:ModifySnapshotAttribute
	ec2:ModifySnapshotTier
	ec2:ModifySpotFleetRequest
	ec2:ModifySubnetAttribute
	ec2:ModifyTrafficMirrorFilterNetworkServices
	ec2:ModifyTrafficMirrorFilterRule
	ec2:ModifyTrafficMirrorSession
	ec2:ModifyTransitGateway
	ec2:ModifyTransitGatewayMeteringPolicy
	ec2:ModifyTransitGatewayPrefixListReference
	ec2:ModifyTransitGatewayVpcAttachment

Service prefix	Actions
	ec2:ModifyVerifiedAccessEndpoint
	ec2:ModifyVerifiedAccessEndpointPolicy
	ec2:ModifyVerifiedAccessGroup
	ec2:ModifyVerifiedAccessGroupPolicy
	ec2:ModifyVerifiedAccessInstance
	ec2:ModifyVerifiedAccessInstanceLoggingConfiguration
	ec2:ModifyVerifiedAccessTrustProvider
	ec2:ModifyVolume
	ec2:ModifyVolumeAttribute
	ec2:ModifyVpcAttribute
	ec2:ModifyVpcBlockPublicAccessExclusion
	ec2:ModifyVpcBlockPublicAccessOptions
	ec2:ModifyVpcEncryptionControl
	ec2:ModifyVpcEndpoint
	ec2:ModifyVpcEndpointConnectionNotification
	ec2:ModifyVpcEndpointServiceConfiguration
	ec2:ModifyVpcEndpointServicePayerResponsibility
	ec2:ModifyVpcEndpointServicePermissions
	ec2:ModifyVpcPeeringConnectionOptions
	ec2:ModifyVpcTenancy
	ec2:ModifyVpnConnection

Service prefix	Actions
	ec2:ModifyVpnConnectionOptions
	ec2:ModifyVpnTunnelCertificate
	ec2:ModifyVpnTunnelOptions
	ec2:MonitorInstances
	ec2:MoveAddressToVpc
	ec2:MoveByoipCidrToIpam
	ec2:MoveCapacityReservationInstances
	ec2:ProvisionByoipCidr
	ec2:ProvisionIpamByoasn
	ec2:ProvisionIpamPoolCidr
	ec2:ProvisionPublicIpv4PoolCidr
	ec2:PurchaseCapacityBlockExtension
	ec2:PurchaseHostReservation
	ec2:PurchaseReservedInstancesOffering
	ec2:PurchaseScheduledInstances
	ec2:RebootInstances
	ec2:RegisterImage
	ec2:RegisterInstanceEventNotificationAttributes
	ec2:RegisterTransitGatewayMulticastGroupMembers
	ec2:RegisterTransitGatewayMulticastGroupSources
	ec2:RejectCapacityReservationBillingOwnership

Service prefix	Actions
	ec2:RejectTransitGatewayClientVpnAttachment
	ec2:RejectTransitGatewayMulticastDomainAssociations
	ec2:RejectTransitGatewayPeeringAttachment
	ec2:RejectTransitGatewayVpcAttachment
	ec2:RejectVpcEndpointConnections
	ec2:RejectVpcPeeringConnection
	ec2:ReleaseAddress
	ec2:ReleaseHosts
	ec2:ReleaseIpamPoolAllocation
	ec2:ReplaceIamInstanceProfileAssociation
	ec2:ReplaceImageCriteriaInAllowedImagesSettings
	ec2:ReplaceNetworkAclAssociation
	ec2:ReplaceNetworkAclEntry
	ec2:ReplaceRoute
	ec2:ReplaceRouteTableAssociation
	ec2:ReplaceTransitGatewayRoute
	ec2:ReplaceVpnTunnel
	ec2:ReportInstanceStatus
	ec2:RequestSpotFleet
	ec2:RequestSpotInstances
	ec2:ResetAddressAttribute

Service prefix	Actions
	ec2:ResetEbsDefaultKmsKeyId
	ec2:ResetFpgaImageAttribute
	ec2:ResetImageAttribute
	ec2:ResetInstanceAttribute
	ec2:ResetNetworkInterfaceAttribute
	ec2:ResetSnapshotAttribute
	ec2:RestoreAddressToClassic
	ec2:RestoreImageFromRecycleBin
	ec2:RestoreManagedPrefixListVersion
	ec2:RestoreSnapshotFromRecycleBin
	ec2:RestoreSnapshotTier
	ec2:RestoreVolumeFromRecycleBin
	ec2:RevokeClientVpnIngress
	ec2:RevokeSecurityGroupEgress
	ec2:RevokeSecurityGroupIngress
	ec2:RunInstances
	ec2:RunScheduledInstances
	ec2:SearchLocalGatewayRoutes
	ec2:SearchTransitGatewayMulticastGroups
	ec2:SearchTransitGatewayRoutes
	ec2:SendDiagnosticInterrupt

Service prefix	Actions
	ec2:StartDeclarativePoliciesReport
	ec2:StartInstances
	ec2:StartNetworkInsightsAccessScopeAnalysis
	ec2:StartNetworkInsightsAnalysis
	ec2:StartVpcEndpointServicePrivateDnsVerification
	ec2:TerminateClientVpnConnections
	ec2:UnassignIpv6Addresses
	ec2:UnassignPrivateIpAddresses
	ec2:UnassignPrivateNatGatewayAddress
	ec2:UnlockSnapshot
	ec2:UnmonitorInstances
	ec2:UpdateCapacityManagerOrganizationsAccess
	ec2:UpdateInterruptibleCapacityReservationAllocation
	ec2:UpdateSecurityGroupRuleDescriptionsEgress
	ec2:UpdateSecurityGroupRuleDescriptionsIngress
	ec2:WithdrawByoipCidr

Service prefix	Actions
ecr	ecr:BatchCheckLayerAvailability ecr:BatchDeleteImage ecr:BatchGetImage ecr:BatchGetRepositoryScanningConfiguration ecr:CompleteLayerUpload ecr:CreatePullThroughCacheRule ecr:CreateRepositoryCreationTemplate ecr:DeleteLifecyclePolicy ecr:DeletePullThroughCacheRule ecr:DeleteRegistryPolicy ecr:DeleteRepository ecr:DeleteRepositoryCreationTemplate ecr:DeleteRepositoryPolicy ecr:DeleteSigningConfiguration ecr:DescribeImageReplicationStatus ecr:DescribeImageScanFindings ecr:DescribeImages ecr:DescribePullThroughCacheRules ecr:DescribeRegistry ecr:DescribeRepositories ecr:DescribeRepositoryCreationTemplates

Service prefix	Actions
	ecr:GetAccountSetting ecr:GetAuthorizationToken ecr:GetDownloadUrlForLayer ecr:GetLifecyclePolicy ecr:GetLifecyclePolicyPreview ecr:GetRegistryPolicy ecr:GetRegistryScanningConfiguration ecr:GetRepositoryPolicy ecr:GetSigningConfiguration ecr:InitiateLayerUpload ecr:ListImages ecr:ListPullTimeUpdateExclusions ecr:PutAccountSetting ecr:PutImage ecr:PutImageScanningConfiguration ecr:PutRegistryPolicy ecr:PutRegistryScanningConfiguration ecr:PutReplicationConfiguration ecr:StartImageScan ecr:StartLifecyclePolicyPreview ecr:UpdatePullThroughCacheRule

Service prefix	Actions
	ecr:UpdateRepositoryCreationTemplate ecr:UploadLayerPart ecr:ValidatePullThroughCacheRule

Service prefix	Actions
ecr-public	ecr-public:BatchCheckLayerAvailability
	ecr-public:BatchDeleteImage
	ecr-public:CompleteLayerUpload
	ecr-public:CreateRepository
	ecr-public:DeleteRepository
	ecr-public:DeleteRepositoryPolicy
	ecr-public:DescribeImages
	ecr-public:DescribeRegistries
	ecr-public:DescribeRepositories
	ecr-public:GetAuthorizationToken
	ecr-public:GetRegistryCatalogData
	ecr-public:GetRepositoryCatalogData
	ecr-public:GetRepositoryPolicy
	ecr-public:InitiateLayerUpload
	ecr-public:PutImage
	ecr-public:PutRegistryCatalogData
	ecr-public:PutRepositoryCatalogData
	ecr-public:SetRepositoryPolicy
	ecr-public:UploadLayerPart

Service prefix	Actions
ecs	ecs:ContinueServiceDeployment ecs:CreateCapacityProvider ecs:CreateCluster ecs:CreateDaemon ecs:CreateService ecs:CreateTaskSet ecs:DeleteAccountSetting ecs:DeleteAttributes ecs:DeleteCapacityProvider ecs:DeleteCluster ecs:DeleteDaemon ecs:DeleteDaemonTaskDefinition ecs:DeleteExpressGatewayService ecs:DeleteService ecs:DeleteTaskDefinitions ecs:DeleteTaskSet ecs:DeregisterContainerInstance ecs:DeregisterTaskDefinition ecs:DescribeCapacityProviders ecs:DescribeClusters ecs:DescribeContainerInstances

Service prefix	Actions
	ecs:DescribeDaemon ecs:DescribeDaemonDeployments ecs:DescribeDaemonRevisions ecs:DescribeDaemonTaskDefinition ecs:DescribeExpressGatewayService ecs:DescribeServiceDeployments ecs:DescribeServiceRevisions ecs:DescribeServices ecs:DescribeTaskDefinition ecs:DescribeTaskSets ecs:DescribeTasks ecs:DiscoverPollEndpoint ecs:ExecuteCommand ecs:GetTaskProtection ecs:ListAccountSettings ecs:ListAttributes ecs:ListClusters ecs:ListContainerInstances ecs:ListDaemonDeployments ecs:ListDaemonTaskDefinitions ecs:ListDaemons

Service prefix	Actions
	ecs:ListServiceDeployments ecs:ListServices ecs:ListServicesByNamespace ecs:ListTaskDefinitionFamilies ecs:ListTaskDefinitions ecs:ListTasks ecs:PutAccountSetting ecs:PutAccountSettingDefault ecs:PutAttributes ecs:PutClusterCapacityProviders ecs:RegisterContainerInstance ecs:RegisterDaemonTaskDefinition ecs:RunTask ecs:StartTask ecs:StopServiceDeployment ecs:StopTask ecs:SubmitAttachmentStateChanges ecs:SubmitContainerStateChange ecs:SubmitTaskStateChange ecs:UpdateCapacityProvider ecs:UpdateCluster

Service prefix	Actions
	ecs:UpdateClusterSettings ecs:UpdateContainerAgent ecs:UpdateContainerInstancesState ecs:UpdateDaemon ecs:UpdateExpressGatewayService ecs:UpdateService ecs:UpdateServicePrimaryTaskSet ecs:UpdateTaskProtection ecs:UpdateTaskSet

Service prefix	Actions
eks	eks:AssociateAccessPolicy
	eks:AssociateEncryptionConfig
	eks:AssociateIdentityProviderConfig
	eks:CancelUpdate
	eks:CreateAccessEntry
	eks:CreateAddon
	eks:CreateCluster
	eks:CreateEksAnywhereSubscription
	eks:CreateFargateProfile
	eks:CreateNodegroup
	eks:DeleteAccessEntry
	eks:DeleteAddon
	eks:DeleteCapability
	eks:DeleteCluster
	eks:DeleteEksAnywhereSubscription
	eks:DeleteFargateProfile
	eks:DeleteNodegroup
	eks:DeletePodIdentityAssociation
	eks:DeregisterCluster
	eks:DescribeAccessEntry
	eks:DescribeAddon

Service prefix	Actions
	eks:DescribeAddonConfiguration
	eks:DescribeAddonVersions
	eks:DescribeCapability
	eks:DescribeCluster
	eks:DescribeClusterVersions
	eks:DescribeEksAnywhereSubscription
	eks:DescribeFargateProfile
	eks:DescribeIdentityProviderConfig
	eks:DescribeInsight
	eks:DescribeInsightsRefresh
	eks:DescribeNodegroup
	eks:DescribePodIdentityAssociation
	eks:DescribeUpdate
	eks:DisassociateAccessPolicy
	eks:DisassociateIdentityProviderConfig
	eks:ListAccessEntries
	eks:ListAccessPolicies
	eks:ListAddons
	eks:ListAssociatedAccessPolicies
	eks:ListCapabilities
	eks:ListClusters

Service prefix	Actions
	eks:ListEksAnywhereSubscriptions
	eks:ListFargateProfiles
	eks:ListIdentityProviderConfigs
	eks:ListInsights
	eks:ListNodegroups
	eks:ListPodIdentityAssociations
	eks:ListUpdates
	eks:RegisterCluster
	eks:StartInsightsRefresh
	eks:UpdateAccessEntry
	eks:UpdateAddon
	eks:UpdateCapability
	eks:UpdateClusterConfig
	eks:UpdateClusterVersion
	eks:UpdateEksAnywhereSubscription
	eks:UpdateNodegroupConfig
	eks:UpdateNodegroupVersion
	eks:UpdatePodIdentityAssociation

Service prefix	Actions
elasticache	elasticache:AuthorizeCacheSecurityGroupIngress elasticache:BatchApplyUpdateAction elasticache:BatchStopUpdateAction elasticache:CompleteMigration elasticache:CopyServerlessCacheSnapshot elasticache:CopySnapshot elasticache:CreateCacheCluster elasticache:CreateCacheParameterGroup elasticache:CreateCacheSecurityGroup elasticache:CreateCacheSubnetGroup elasticache:CreateGlobalReplicationGroup elasticache:CreateReplicationGroup elasticache:CreateServerlessCache elasticache:CreateServerlessCacheSnapshot elasticache:CreateSnapshot elasticache:CreateUser elasticache:CreateUserGroup elasticache:DecreaseNodeGroupsInGlobalReplicationGroup elasticache:DecreaseReplicaCount elasticache:DeleteCacheCluster elasticache:DeleteCacheParameterGroup

Service prefix	Actions
	elasticache:DeleteCacheSecurityGroup elasticache:DeleteCacheSubnetGroup elasticache:DeleteGlobalReplicationGroup elasticache:DeleteReplicationGroup elasticache:DeleteServerlessCache elasticache:DeleteServerlessCacheSnapshot elasticache:DeleteSnapshot elasticache:DeleteUser elasticache:DeleteUserGroup elasticache:DescribeCacheClusters elasticache:DescribeCacheEngineVersions elasticache:DescribeCacheParameterGroups elasticache:DescribeCacheParameters elasticache:DescribeCacheSecurityGroups elasticache:DescribeCacheSubnetGroups elasticache:DescribeEngineDefaultParameters elasticache:DescribeEvents elasticache:DescribeGlobalReplicationGroups elasticache:DescribeReplicationGroups elasticache:DescribeReservedCacheNodes elasticache:DescribeReservedCacheNodesOfferings

Service prefix	Actions
	elasticache:DescribeServerlessCacheSnapshots elasticache:DescribeServerlessCaches elasticache:DescribeServiceUpdates elasticache:DescribeSnapshots elasticache:DescribeUpdateActions elasticache:DescribeUserGroups elasticache:DescribeUsers elasticache:DisassociateGlobalReplicationGroup elasticache:ExportServerlessCacheSnapshot elasticache:FailoverGlobalReplicationGroup elasticache:IncreaseNodeGroupsInGlobalReplicationGroup elasticache:IncreaseReplicaCount elasticache:ListAllowedNodeTypeModifications elasticache:ModifyCacheCluster elasticache:ModifyCacheParameterGroup elasticache:ModifyCacheSubnetGroup elasticache:ModifyGlobalReplicationGroup elasticache:ModifyReplicationGroup elasticache:ModifyReplicationGroupShardConfiguration elasticache:ModifyServerlessCache elasticache:ModifyUser

Service prefix	Actions
	elasticache:ModifyUserGroup elasticache:PurchaseReservedCacheNodesOffering elasticache:RebalanceSlotsInGlobalReplicationGroup elasticache:RebootCacheCluster elasticache:ResetCacheParameterGroup elasticache:RevokeCacheSecurityGroupIngress elasticache:StartMigration elasticache:TestFailover elasticache:TestMigration

Service prefix	Actions
elasticbeanstalk	elasticbeanstalk:AbortEnvironmentUpdate
	elasticbeanstalk:ApplyEnvironmentManagedAction
	elasticbeanstalk:AssociateEnvironmentOperationsRole
	elasticbeanstalk:CheckDNSAvailability
	elasticbeanstalk:ComposeEnvironments
	elasticbeanstalk:CreateApplication
	elasticbeanstalk:CreateApplicationVersion
	elasticbeanstalk:CreateConfigurationTemplate
	elasticbeanstalk:CreateEnvironment
	elasticbeanstalk:CreatePlatformVersion
	elasticbeanstalk:CreateStorageLocation
	elasticbeanstalk:DeleteApplication
	elasticbeanstalk:DeleteApplicationVersion
	elasticbeanstalk:DeleteConfigurationTemplate
	elasticbeanstalk:DeleteEnvironmentConfiguration
	elasticbeanstalk:DeletePlatformVersion
	elasticbeanstalk:DescribeAccountAttributes
	elasticbeanstalk:DescribeApplicationVersions
	elasticbeanstalk:DescribeApplications
	elasticbeanstalk:DescribeConfigurationOptions
	elasticbeanstalk:DescribeConfigurationSettings

Service prefix	Actions
	elasticbeanstalk:DescribeEnvironmentHealth
	elasticbeanstalk:DescribeEnvironmentManagedActionHistory
	elasticbeanstalk:DescribeEnvironmentManagedActions
	elasticbeanstalk:DescribeEnvironmentResources
	elasticbeanstalk:DescribeEnvironments
	elasticbeanstalk:DescribeEvents
	elasticbeanstalk:DescribeInstancesHealth
	elasticbeanstalk:DescribePlatformVersion
	elasticbeanstalk:DisassociateEnvironmentOperationsRole
	elasticbeanstalk:ListAvailableSolutionStacks
	elasticbeanstalk:ListPlatformBranches
	elasticbeanstalk:ListPlatformVersions
	elasticbeanstalk:RebuildEnvironment
	elasticbeanstalk:RequestEnvironmentInfo
	elasticbeanstalk:RestartAppServer
	elasticbeanstalk:RetrieveEnvironmentInfo
	elasticbeanstalk:SwapEnvironmentCNAMEs
	elasticbeanstalk:TerminateEnvironment
	elasticbeanstalk:UpdateApplication
	elasticbeanstalk:UpdateApplicationResourceLifecycle
	elasticbeanstalk:UpdateApplicationVersion

Service prefix	Actions
	elasticbeanstalk:UpdateConfigurationTemplate elasticbeanstalk:UpdateEnvironment elasticbeanstalk:ValidateConfigurationSettings

Service prefix	Actions
elasticfilesystem	elasticfilesystem:CreateAccessPoint elasticfilesystem:CreateFileSystem elasticfilesystem:CreateMountTarget elasticfilesystem:CreateReplicationConfiguration elasticfilesystem:DeleteAccessPoint elasticfilesystem:DeleteFileSystem elasticfilesystem:DeleteFileSystemPolicy elasticfilesystem:DeleteMountTarget elasticfilesystem:DeleteReplicationConfiguration elasticfilesystem:DescribeAccessPoints elasticfilesystem:DescribeAccountPreferences elasticfilesystem:DescribeBackupPolicy elasticfilesystem:DescribeFileSystemPolicy elasticfilesystem:DescribeFileSystems elasticfilesystem:DescribeLifecycleConfiguration elasticfilesystem:DescribeMountTargetSecurityGroups elasticfilesystem:DescribeMountTargets elasticfilesystem:DescribeReplicationConfigurations elasticfilesystem:ModifyMountTargetSecurityGroups elasticfilesystem:PutAccountPreferences elasticfilesystem:PutBackupPolicy

Service prefix	Actions
	elasticfilesystem:PutFileSystemPolicy elasticfilesystem:PutLifecycleConfiguration elasticfilesystem:UpdateFileSystem elasticfilesystem:UpdateFileSystemProtection

Service prefix	Actions
elasticloadbalancing	elasticloadbalancing:AddListenerCertificates
	elasticloadbalancing:AddTrustStoreRevocations
	elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
	elasticloadbalancing:AttachLoadBalancerToSubnets
	elasticloadbalancing:ConfigureHealthCheck
	elasticloadbalancing:CreateAppCookieStickinessPolicy
	elasticloadbalancing:CreateLBCookieStickinessPolicy
	elasticloadbalancing:CreateListener
	elasticloadbalancing:CreateLoadBalancer
	elasticloadbalancing:CreateLoadBalancerListeners
	elasticloadbalancing:CreateLoadBalancerPolicy
	elasticloadbalancing:CreateRule
	elasticloadbalancing:CreateTargetGroup
	elasticloadbalancing:CreateTrustStore
	elasticloadbalancing:DeleteListener
	elasticloadbalancing:DeleteLoadBalancer
	elasticloadbalancing:DeleteLoadBalancerListeners
	elasticloadbalancing:DeleteLoadBalancerPolicy
	elasticloadbalancing:DeleteRule
	elasticloadbalancing:DeleteSharedTrustStoreAssociation
	elasticloadbalancing:DeleteTargetGroup

Service prefix	Actions
	elasticloadbalancing:DeleteTrustStore
	elasticloadbalancing:DeregisterInstancesFromLoadBalancer
	elasticloadbalancing:DeregisterTargets
	elasticloadbalancing:DescribeAccountLimits
	elasticloadbalancing:DescribeCapacityReservation
	elasticloadbalancing:DescribeInstanceHealth
	elasticloadbalancing:DescribeListenerAttributes
	elasticloadbalancing:DescribeListenerCertificates
	elasticloadbalancing:DescribeListeners
	elasticloadbalancing:DescribeLoadBalancerAttributes
	elasticloadbalancing:DescribeLoadBalancerPolicies
	elasticloadbalancing:DescribeLoadBalancerPolicyTypes
	elasticloadbalancing:DescribeLoadBalancers
	elasticloadbalancing:DescribeRules
	elasticloadbalancing:DescribeSSLPolicies
	elasticloadbalancing:DescribeTargetGroupAttributes
	elasticloadbalancing:DescribeTargetGroups
	elasticloadbalancing:DescribeTargetHealth
	elasticloadbalancing:DescribeTrustStoreAssociations
	elasticloadbalancing:DescribeTrustStoreRevocations
	elasticloadbalancing:DescribeTrustStores

Service prefix	Actions
	elasticloadbalancing:DetachLoadBalancerFromSubnets
	elasticloadbalancing:DisableAvailabilityZonesForLoadBalancer
	elasticloadbalancing:EnableAvailabilityZonesForLoadBalancer
	elasticloadbalancing:GetResourcePolicy
	elasticloadbalancing:GetTrustStoreCaCertificatesBundle
	elasticloadbalancing:GetTrustStoreRevocationContent
	elasticloadbalancing:ModifyCapacityReservation
	elasticloadbalancing:ModifyIpPools
	elasticloadbalancing:ModifyListener
	elasticloadbalancing:ModifyLoadBalancerAttributes
	elasticloadbalancing:ModifyRule
	elasticloadbalancing:ModifyTargetGroup
	elasticloadbalancing:ModifyTargetGroupAttributes
	elasticloadbalancing:ModifyTrustStore
	elasticloadbalancing:RegisterInstancesWithLoadBalancer
	elasticloadbalancing:RegisterTargets
	elasticloadbalancing:RemoveListenerCertificates
	elasticloadbalancing:RemoveTrustStoreRevocations
	elasticloadbalancing:SetIpAddressType
	elasticloadbalancing:SetLoadBalancerListenerSSLCertificate
	elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer

Service prefix	Actions
	elasticloadbalancing:SetLoadBalancerPoliciesOfListener elasticloadbalancing:SetRulePriorities elasticloadbalancing:SetSecurityGroups elasticloadbalancing:SetSubnets

Service prefix	Actions
emr-containers	emr-containers:CancelJobRun emr-containers:CreateJobTemplate emr-containers:CreateManagedEndpoint emr-containers:CreateSecurityConfiguration emr-containers:CreateVirtualCluster emr-containers:DeleteJobTemplate emr-containers:DeleteManagedEndpoint emr-containers:DeleteSecurityConfiguration emr-containers:DeleteVirtualCluster emr-containers:DescribeJobRun emr-containers:DescribeJobTemplate emr-containers:DescribeManagedEndpoint emr-containers:DescribeSecurityConfiguration emr-containers:DescribeVirtualCluster emr-containers:GetManagedEndpointSessionCredentials emr-containers:ListJobRuns emr-containers:ListJobTemplates emr-containers:ListManagedEndpoints emr-containers:ListSecurityConfigurations emr-containers:ListVirtualClusters emr-containers:StartJobRun

Service prefix	Actions
emr-serverless	emr-serverless:CancelJobRun
	emr-serverless:CreateApplication
	emr-serverless:DeleteApplication
	emr-serverless:GetApplication
	emr-serverless:GetDashboardForJobRun
	emr-serverless:GetJobRun
	emr-serverless:GetResourceDashboard
	emr-serverless:GetSession
	emr-serverless:GetSessionEndpoint
	emr-serverless:ListApplications
	emr-serverless:ListJobRunAttempts
	emr-serverless:ListJobRuns
	emr-serverless:ListSessions
	emr-serverless:StartApplication
	emr-serverless:StartJobRun
	emr-serverless:StopApplication
	emr-serverless:TerminateSession
	emr-serverless:UpdateApplication

Service prefix	Actions
es	es:AcceptInboundConnection es:AcceptInboundCrossClusterSearchConnection es:AssociatePackage es:AuthorizeVpcEndpointAccess es:CancelElasticsearchServiceSoftwareUpdate es:CancelServiceSoftwareUpdate es:CreateDomain es:CreateElasticsearchDomain es:CreateIndex es:CreateOutboundConnection es:CreateOutboundCrossClusterSearchConnection es:CreatePackage es:CreateVpcEndpoint es:DeleteDomain es:DeleteElasticsearchDomain es:DeleteElasticsearchServiceRole es:DeleteInboundConnection es:DeleteInboundCrossClusterSearchConnection es:DeleteIndex es:DeleteOutboundConnection es:DeleteOutboundCrossClusterSearchConnection

Service prefix	Actions
	es:DeletePackage es:DeleteVpcEndpoint es:DescribeDomain es:DescribeDomainAutoTunes es:DescribeDomainChangeProgress es:DescribeDomainConfig es:DescribeDomainHealth es:DescribeDomainNodes es:DescribeDomains es:DescribeDryRunProgress es:DescribeElasticsearchDomain es:DescribeElasticsearchDomainConfig es:DescribeElasticsearchDomains es:DescribeElasticsearchInstanceTypeLimits es:DescribeInboundConnections es:DescribeInboundCrossClusterSearchConnections es:DescribeInstanceTypeLimits es:DescribeOutboundConnections es:DescribeOutboundCrossClusterSearchConnections es:DescribePackages es:DescribeReservedElasticsearchInstanceOfferings

Service prefix	Actions
	es:DescribeReservedElasticsearchInstances es:DescribeReservedInstanceOfferings es:DescribeReservedInstances es:DescribeVpcEndpoints es:DissociatePackage es:DissociatePackages es:GetCompatibleElasticsearchVersions es:GetCompatibleVersions es:GetDataSource es:GetDomainMaintenanceStatus es:GetPackageVersionHistory es:GetUpgradeHistory es:GetUpgradeStatus es:ListDataSources es:ListDomainNames es:ListDomainsForPackage es:ListElasticsearchInstanceTypes es:ListElasticsearchVersions es:ListInsights es:ListInstanceTypeDetails es:ListPackagesForDomain

Service prefix	Actions
	es:ListScheduledActions es:ListVersions es:ListVpcEndpointAccess es:ListVpcEndpoints es:ListVpcEndpointsForDomain es:PurchaseReservedElasticsearchInstanceOffering es:PurchaseReservedInstanceOffering es:RejectInboundConnection es:RejectInboundCrossClusterSearchConnection es:RevokeVpcEndpointAccess es:RollbackServiceSoftwareUpdate es:StartDomainMaintenance es:StartElasticsearchServiceSoftwareUpdate es:StartServiceSoftwareUpdate es:UpdateDataSource es:UpdateDomainConfig es:UpdateElasticsearchDomainConfig es:UpdateIndex es:UpdatePackage es:UpdatePackageScope es:UpdateScheduledAction

Service prefix	Actions
	es:UpdateVpcEndpoint es:UpgradeDomain es:UpgradeElasticsearchDomain

Service prefix	Actions
events	events:ActivateEventSource
	events:CancelReplay
	events:CreateApiDestination
	events:CreateArchive
	events:CreateConnection
	events:CreateEndpoint
	events:CreateEventBus
	events:CreatePartnerEventSource
	events:DeactivateEventSource
	events:DeauthorizeConnection
	events:DeleteApiDestination
	events:DeleteArchive
	events:DeleteConnection
	events:DeleteEndpoint
	events:DeleteEventBus
	events:DeletePartnerEventSource
	events:DeleteRule
	events:DescribeApiDestination
	events:DescribeArchive
	events:DescribeConnection
	events:DescribeEndpoint

Service prefix	Actions
	events:DescribeEventBus
	events:DescribeEventSource
	events:DescribePartnerEventSource
	events:DescribeReplay
	events:DescribeRule
	events:DisableRule
	events:EnableRule
	events:ListApiDestinations
	events:ListArchives
	events:ListConnections
	events:ListEndpoints
	events:ListEventBuses
	events:ListEventSources
	events:ListPartnerEventSourceAccounts
	events:ListPartnerEventSources
	events:ListReplays
	events:ListRuleNamesByTarget
	events:ListRules
	events:ListTargetsByRule
	events:PutPermission
	events:PutRule

Service prefix	Actions
	events:PutTargets
	events:RemovePermission
	events:RemoveTargets
	events:StartReplay
	events:TestEventPattern
	events:UpdateApiDestination
	events:UpdateArchive
	events:UpdateConnection
	events:UpdateEndpoint
	events:UpdateEventBus

Service prefix	Actions
finspace	finspace:CreateEnvironment finspace:CreateKxChangeset finspace:CreateKxCluster finspace:CreateKxDatabase finspace:CreateKxDataview finspace:CreateKxEnvironment finspace:CreateKxScalingGroup finspace:CreateKxUser finspace:CreateKxVolume finspace:CreateUser finspace:DeleteEnvironment finspace:DeleteKxCluster finspace:DeleteKxClusterNode finspace:DeleteKxDatabase finspace:DeleteKxDataview finspace:DeleteKxEnvironment finspace:DeleteKxScalingGroup finspace:DeleteKxUser finspace:DeleteKxVolume finspace:GetEnvironment finspace:GetKxChangeset

Service prefix	Actions
	finspace:GetKxCluster finspace:GetKxConnectionString finspace:GetKxDatabase finspace:GetKxDataview finspace:GetKxEnvironment finspace:GetKxScalingGroup finspace:GetKxUser finspace:GetKxVolume finspace:GetLoadSampleDataSetGroupIntoEnvironmentStatus finspace:GetUser finspace:ListEnvironments finspace:ListKxChangesets finspace:ListKxClusterNodes finspace:ListKxClusters finspace:ListKxDatabases finspace:ListKxDataviews finspace:ListKxEnvironments finspace:ListKxScalingGroups finspace:ListKxUsers finspace:ListKxVolumes finspace:ListUsers

Service prefix	Actions
	finspace:LoadSampleDataSetGroupIntoEnvironment finspace:ResetUserPassword finspace:UpdateEnvironment finspace:UpdateKxClusterCodeConfiguration finspace:UpdateKxClusterDatabases finspace:UpdateKxDatabase finspace:UpdateKxDataview finspace:UpdateKxEnvironment finspace:UpdateKxEnvironmentNetwork finspace:UpdateKxUser finspace:UpdateKxVolume finspace:UpdateUser
firehose	firehose:CreateDeliveryStream firehose:DeleteDeliveryStream firehose:DescribeDeliveryStream firehose:ListDeliveryStreams firehose:StartDeliveryStreamEncryption firehose:StopDeliveryStreamEncryption firehose:UpdateDestination

Service prefix	Actions
fis	fis:CreateExperimentTemplate fis:CreateTargetAccountConfiguration fis:DeleteExperimentTemplate fis:DeleteTargetAccountConfiguration fis:GetAction fis:GetExperiment fis:GetExperimentTargetAccountConfiguration fis:GetExperimentTemplate fis:GetSafetyLever fis:GetTargetAccountConfiguration fis:GetTargetResourceType fis:ListActions fis:ListExperimentResolvedTargets fis:ListExperimentTargetAccountConfigurations fis:ListExperimentTemplates fis:ListExperiments fis:ListTargetAccountConfigurations fis:ListTargetResourceTypes fis:StartExperiment fis:StopExperiment fis:UpdateExperimentTemplate

Service prefix	Actions
	fis:UpdateSafetyLeverState fis:UpdateTargetAccountConfiguration

Service prefix	Actions
fms	fms:AssociateAdminAccount
	fms:AssociateThirdPartyFirewall
	fms:BatchAssociateResource
	fms:BatchDisassociateResource
	fms:DeleteAppsList
	fms:DeleteNotificationChannel
	fms:DeletePolicy
	fms:DeleteProtocolsList
	fms:DeleteResourceSet
	fms:DisassociateAdminAccount
	fms:DisassociateThirdPartyFirewall
	fms:GetAdminAccount
	fms:GetAdminScope
	fms:GetAppsList
	fms:GetComplianceDetail
	fms:GetNotificationChannel
	fms:GetPolicy
	fms:GetProtectionStatus
	fms:GetProtocolsList
	fms:GetResourceSet
	fms:GetThirdPartyFirewallAssociationStatus

Service prefix	Actions
	fms:GetViolationDetails
	fms:ListAdminAccountsForOrganization
	fms:ListAdminsManagingAccount
	fms:ListAppsLists
	fms:ListComplianceStatus
	fms:ListDiscoveredResources
	fms:ListMemberAccounts
	fms:ListPolicies
	fms:ListProtocolsLists
	fms:ListResourceSetResources
	fms:ListResourceSets
	fms:ListThirdPartyFirewallFirewallPolicies
	fms:PutAdminAccount
	fms:PutAppsList
	fms:PutNotificationChannel
	fms:PutPolicy
	fms:PutProtocolsList
	fms:PutResourceSet

Service prefix	Actions
frauddetector	frauddetector:BatchCreateVariable frauddetector:BatchGetVariable frauddetector:CancelBatchImportJob frauddetector:CancelBatchPredictionJob frauddetector:CreateBatchImportJob frauddetector:CreateBatchPredictionJob frauddetector:CreateDetectorVersion frauddetector:CreateList frauddetector:CreateModel frauddetector:CreateModelVersion frauddetector:CreateRule frauddetector:CreateVariable frauddetector:DeleteBatchImportJob frauddetector:DeleteBatchPredictionJob frauddetector:DeleteDetector frauddetector:DeleteDetectorVersion frauddetector:DeleteEntityType frauddetector:DeleteEvent frauddetector:DeleteEventType frauddetector:DeleteEventsByEventType frauddetector:DeleteExternalModel

Service prefix	Actions
	frauddetector:DeleteLabel frauddetector:DeleteList frauddetector:DeleteModel frauddetector:DeleteModelVersion frauddetector:DeleteOutcome frauddetector:DeleteRule frauddetector:DeleteVariable frauddetector:DescribeDetector frauddetector:DescribeModelVersions frauddetector:GetBatchImportJobs frauddetector:GetBatchPredictionJobs frauddetector:GetDeleteEventsByEventTypeStatus frauddetector:GetDetectorVersion frauddetector:GetDetectors frauddetector:GetEntityTypes frauddetector:GetEvent frauddetector:GetEventPrediction frauddetector:GetEventPredictionMetadata frauddetector:GetEventTypes frauddetector:GetExternalModels frauddetector:GetKMSEncryptionKey

Service prefix	Actions
	frauddetector:GetLabels frauddetector:GetListElements frauddetector:GetListsMetadata frauddetector:GetModelVersion frauddetector:GetModels frauddetector:GetOutcomes frauddetector:GetRules frauddetector:GetVariables frauddetector:ListEventPredictions frauddetector:PutDetector frauddetector:PutEntityType frauddetector:PutEventType frauddetector:PutExternalModel frauddetector:PutKMSEncryptionKey frauddetector:PutLabel frauddetector:PutOutcome frauddetector:SendEvent frauddetector:UpdateDetectorVersion frauddetector:UpdateDetectorVersionMetadata frauddetector:UpdateDetectorVersionStatus frauddetector:UpdateEventLabel

Service prefix	Actions
	frauddetector:UpdateList frauddetector:UpdateModel frauddetector:UpdateModelVersion frauddetector:UpdateModelVersionStatus frauddetector:UpdateRuleMetadata frauddetector:UpdateRuleVersion frauddetector:UpdateVariable

Service prefix	Actions
fsx	fsx:AssociateFileSystemAliases
	fsx:CancelDataRepositoryTask
	fsx:CopyBackup
	fsx:CreateDataRepositoryTask
	fsx:CreateFileCache
	fsx:CreateFileSystem
	fsx:CreateFileSystemFromBackup
	fsx:CreateSnapshot
	fsx:CreateStorageVirtualMachine
	fsx:CreateVolume
	fsx:CreateVolumeFromBackup
	fsx:DeleteBackup
	fsx:DeleteFileCache
	fsx:DeleteFileSystem
	fsx:DeleteSnapshot
	fsx:DeleteStorageVirtualMachine
	fsx:DeleteVolume
	fsx:DescribeBackups
	fsx:DescribeDataRepositoryAssociations
	fsx:DescribeDataRepositoryTasks
	fsx:DescribeFileCaches

Service prefix	Actions
	fsx:DescribeFileSystemAliases
	fsx:DescribeFileSystems
	fsx:DescribeS3AccessPointAttachments
	fsx:DescribeSharedVpcConfiguration
	fsx:DescribeSnapshots
	fsx:DescribeStorageVirtualMachines
	fsx:DescribeVolumes
	fsx:DetachAndDeleteS3AccessPoint
	fsx:DisassociateFileSystemAliases
	fsx:ReleaseFileSystemNfsV3Locks
	fsx:RestoreVolumeFromSnapshot
	fsx:StartMisconfiguredStateRecovery
	fsx:UpdateDataRepositoryAssociation
	fsx:UpdateFileCache
	fsx:UpdateFileSystem
	fsx:UpdateSharedVpcConfiguration
	fsx:UpdateSnapshot
	fsx:UpdateStorageVirtualMachine
	fsx:UpdateVolume

Service prefix	Actions
gamelift	gamelift:AcceptMatch gamelift:ClaimGameServer gamelift:CreateAlias gamelift:CreateBuild gamelift:CreateContainerGroupDefinition gamelift:CreateFleet gamelift:CreateFleetLocations gamelift:CreateGameServerGroup gamelift:CreateGameSession gamelift:CreateGameSessionQueue gamelift:CreateLocation gamelift:CreateMatchmakingConfiguration gamelift:CreateMatchmakingRuleSet gamelift:CreatePlayerSession gamelift:CreatePlayerSessions gamelift:CreateScript gamelift:CreateVpcPeeringAuthorization gamelift:CreateVpcPeeringConnection gamelift:DeleteAlias gamelift:DeleteBuild gamelift:DeleteContainerGroupDefinition

Service prefix	Actions
	gamelift:DeleteFleet gamelift:DeleteFleetLocations gamelift:DeleteGameServerGroup gamelift:DeleteGameSessionQueue gamelift:DeleteLocation gamelift:DeleteMatchmakingConfiguration gamelift:DeleteMatchmakingRuleSet gamelift:DeleteScalingPolicy gamelift:DeleteScript gamelift:DeleteVpcPeeringAuthorization gamelift:DeleteVpcPeeringConnection gamelift:DeregisterCompute gamelift:DeregisterGameServer gamelift:DescribeAlias gamelift:DescribeBuild gamelift:DescribeCompute gamelift:DescribeContainerFleet gamelift:DescribeContainerGroupDefinition gamelift:DescribeContainerGroupPortMappings gamelift:DescribeEC2InstanceLimits gamelift:DescribeFleetAttributes

Service prefix	Actions
	gamelift:DescribeFleetCapacity gamelift:DescribeFleetEvents gamelift:DescribeFleetLocationAttributes gamelift:DescribeFleetLocationCapacity gamelift:DescribeFleetLocationUtilization gamelift:DescribeFleetPortSettings gamelift:DescribeFleetUtilization gamelift:DescribeGameServer gamelift:DescribeGameServerGroup gamelift:DescribeGameServerInstances gamelift:DescribeGameSessionDetails gamelift:DescribeGameSessionPlacement gamelift:DescribeGameSessionQueues gamelift:DescribeGameSessions gamelift:DescribeInstances gamelift:DescribeMatchmaking gamelift:DescribeMatchmakingConfigurations gamelift:DescribeMatchmakingRuleSets gamelift:DescribePlayerSessions gamelift:DescribeRuntimeConfiguration gamelift:DescribeScalingPolicies

Service prefix	Actions
	gamelift:DescribeScript gamelift:DescribeVpcPeeringAuthorizations gamelift:DescribeVpcPeeringConnections gamelift:GetComputeAccess gamelift:GetComputeAuthToken gamelift:GetGameSessionLogUrl gamelift:GetInstanceAccess gamelift:GetPlayerConnectionDetails gamelift:ListAliases gamelift:ListBuilds gamelift:ListCompute gamelift:ListContainerFleets gamelift:ListContainerGroupDefinitionVersions gamelift:ListContainerGroupDefinitions gamelift:ListFleetDeployments gamelift:ListFleets gamelift:ListGameServerGroups gamelift:ListGameServers gamelift:ListLocations gamelift:ListScripts gamelift:PutScalingPolicy

Service prefix	Actions
	gamelift:RegisterCompute gamelift:RegisterGameServer gamelift:RequestUploadCredentials gamelift:ResolveAlias gamelift:ResumeGameServerGroup gamelift:SearchGameSessions gamelift:StartFleetActions gamelift:StartGameSessionPlacement gamelift:StartMatchBackfill gamelift:StartMatchmaking gamelift:StopFleetActions gamelift:StopGameSessionPlacement gamelift:StopMatchmaking gamelift:SuspendGameServerGroup gamelift:TerminateGameSession gamelift:UpdateAlias gamelift:UpdateBuild gamelift:UpdateContainerGroupDefinition gamelift:UpdateFleetAttributes gamelift:UpdateFleetCapacity gamelift:UpdateFleetPortSettings

Service prefix	Actions
	gamelift:UpdateGameServer gamelift:UpdateGameServerGroup gamelift:UpdateGameSession gamelift:UpdateGameSessionQueue gamelift:UpdateMatchmakingConfiguration gamelift:UpdateRuntimeConfiguration gamelift:UpdateScript gamelift:ValidateMatchmakingRuleSet

Service prefix	Actions
geo	geo:AssociateTrackerConsumer
	geo:BatchDeleteDevicePositionHistory
	geo:BatchDeleteGeofence
	geo:BatchEvaluateGeofences
	geo:BatchGetDevicePosition
	geo:BatchPutGeofence
	geo:BatchUpdateDevicePosition
	geo:CalculateRoute
	geo:CalculateRouteMatrix
	geo:CancelJob
	geo:CreateGeofenceCollection
	geo:CreateMap
	geo:CreatePlaceIndex
	geo:CreateRouteCalculator
	geo:CreateTracker
	geo:DeleteGeofenceCollection
	geo:DeleteKey
	geo:DeleteMap
	geo:DeletePlaceIndex
	geo:DeleteRouteCalculator
	geo:DeleteTracker

Service prefix	Actions
	geo:DescribeGeofenceCollection
	geo:DescribeKey
	geo:DescribeMap
	geo:DescribePlaceIndex
	geo:DescribeRouteCalculator
	geo:DescribeTracker
	geo:DisassociateTrackerConsumer
	geo:ForecastGeofenceEvents
	geo:GetDevicePosition
	geo:GetDevicePositionHistory
	geo:GetGeofence
	geo:GetJob
	geo:GetMapGlyphs
	geo:GetMapSprites
	geo:GetMapStyleDescriptor
	geo:GetMapTile
	geo:GetPlace
	geo:ListDevicePositions
	geo:ListGeofenceCollections
	geo:ListGeofences
	geo:ListJobs

Service prefix	Actions
	geo:ListKeys
	geo:ListMaps
	geo:ListPlaceIndexes
	geo:ListRouteCalculators
	geo:ListTrackerConsumers
	geo:ListTrackers
	geo:PutGeofence
	geo:SearchPlaceIndexForPosition
	geo:SearchPlaceIndexForSuggestions
	geo:SearchPlaceIndexForText
	geo:UpdateGeofenceCollection
	geo:UpdateKey
	geo:UpdateMap
	geo:UpdatePlaceIndex
	geo:UpdateRouteCalculator
	geo:UpdateTracker
	geo:VerifyDevicePosition

Service prefix	Actions
glacier	glacier:AbortMultipartUpload glacier:AbortVaultLock glacier:CompleteMultipartUpload glacier:CompleteVaultLock glacier:CreateVault glacier:DeleteArchive glacier:DeleteVault glacier:DeleteVaultAccessPolicy glacier:DeleteVaultNotifications glacier:DescribeJob glacier:DescribeVault glacier:GetDataRetrievalPolicy glacier:GetJobOutput glacier:GetVaultAccessPolicy glacier:GetVaultLock glacier:GetVaultNotifications glacier:InitiateJob glacier:InitiateMultipartUpload glacier:InitiateVaultLock glacier:ListJobs glacier:ListMultipartUploads

Service prefix	Actions
	glacier:ListParts glacier:ListProvisionedCapacity glacier:ListVaults glacier:PurchaseProvisionedCapacity glacier:SetDataRetrievalPolicy glacier:SetVaultAccessPolicy glacier:SetVaultNotifications glacier:UploadArchive glacier:UploadMultipartPart

Service prefix	Actions
grafana	grafana:AssociateLicense
	grafana:CreateWorkspace
	grafana:CreateWorkspaceApiKey
	grafana:CreateWorkspaceServiceAccount
	grafana:CreateWorkspaceServiceAccountToken
	grafana:DeleteWorkspace
	grafana:DeleteWorkspaceApiKey
	grafana:DeleteWorkspaceServiceAccount
	grafana:DeleteWorkspaceServiceAccountToken
	grafana:DescribeWorkspace
	grafana:DescribeWorkspaceAuthentication
	grafana:DescribeWorkspaceConfiguration
	grafana:DisassociateLicense
	grafana:ListPermissions
	grafana:ListVersions
	grafana:ListWorkspaceServiceAccountTokens
	grafana:ListWorkspaceServiceAccounts
	grafana:ListWorkspaces
	grafana:UpdatePermissions
	grafana:UpdateWorkspace
	grafana:UpdateWorkspaceAuthentication

Service prefix	Actions
	grafana:UpdateWorkspaceConfiguration

Service prefix	Actions
greengrass	greengrass:AssociateRoleToGroup greengrass:AssociateServiceRoleToAccount greengrass:BatchAssociateClientDeviceWithCoreDevice greengrass:BatchDisassociateClientDeviceFromCoreDevice greengrass:CancelDeployment greengrass:CreateComponentVersion greengrass:CreateConnectorDefinition greengrass:CreateConnectorDefinitionVersion greengrass:CreateCoreDefinition greengrass:CreateCoreDefinitionVersion greengrass:CreateDeployment greengrass:CreateDeviceDefinition greengrass:CreateDeviceDefinitionVersion greengrass:CreateFunctionDefinition greengrass:CreateFunctionDefinitionVersion greengrass:CreateGroup greengrass:CreateGroupCertificateAuthority greengrass:CreateGroupVersion greengrass:CreateLoggerDefinition greengrass:CreateLoggerDefinitionVersion greengrass:CreateResourceDefinition

Service prefix	Actions
	greengrass:CreateResourceDefinitionVersion greengrass:CreateSoftwareUpdateJob greengrass:CreateSubscriptionDefinition greengrass:CreateSubscriptionDefinitionVersion greengrass:DeleteComponent greengrass:DeleteConnectorDefinition greengrass:DeleteCoreDefinition greengrass:DeleteCoreDevice greengrass:DeleteDeployment greengrass:DeleteDeviceDefinition greengrass:DeleteFunctionDefinition greengrass:DeleteGroup greengrass:DeleteLoggerDefinition greengrass:DeleteResourceDefinition greengrass:DeleteSubscriptionDefinition greengrass:DescribeComponent greengrass:DisassociateRoleFromGroup greengrass:DisassociateServiceRoleFromAccount greengrass:GetAssociatedRole greengrass:GetBulkDeploymentStatus greengrass:GetComponent

Service prefix	Actions
	greengrass:GetComponentVersionArtifact greengrass:GetConnectivityInfo greengrass:GetConnectorDefinition greengrass:GetConnectorDefinitionVersion greengrass:GetCoreDefinition greengrass:GetCoreDefinitionVersion greengrass:GetCoreDevice greengrass:GetDeployment greengrass:GetDeploymentStatus greengrass:GetDeviceDefinition greengrass:GetDeviceDefinitionVersion greengrass:GetFunctionDefinition greengrass:GetFunctionDefinitionVersion greengrass:GetGroup greengrass:GetGroupCertificateAuthority greengrass:GetGroupCertificateConfiguration greengrass:GetGroupVersion greengrass:GetLoggerDefinition greengrass:GetLoggerDefinitionVersion greengrass:GetResourceDefinition greengrass:GetResourceDefinitionVersion

Service prefix	Actions
	greengrass:GetServiceRoleForAccount greengrass:GetSubscriptionDefinition greengrass:GetSubscriptionDefinitionVersion greengrass:GetThingRuntimeConfiguration greengrass:ListBulkDeploymentDetailedReports greengrass:ListBulkDeployments greengrass:ListClientDevicesAssociatedWithCoreDevice greengrass:ListComponentVersions greengrass:ListComponents greengrass:ListConnectorDefinitionVersions greengrass:ListConnectorDefinitions greengrass:ListCoreDefinitionVersions greengrass:ListCoreDefinitions greengrass:ListCoreDevices greengrass:ListDeployments greengrass:ListDeviceDefinitionVersions greengrass:ListDeviceDefinitions greengrass:ListEffectiveDeployments greengrass:ListFunctionDefinitionVersions greengrass:ListFunctionDefinitions greengrass:ListGroupCertificateAuthorities

Service prefix	Actions
	greengrass:ListGroupVersions greengrass:ListGroups greengrass:ListInstalledComponents greengrass:ListLoggerDefinitionVersions greengrass:ListLoggerDefinitions greengrass:ListResourceDefinitionVersions greengrass:ListResourceDefinitions greengrass:ListSubscriptionDefinitionVersions greengrass:ListSubscriptionDefinitions greengrass:ResetDeployments greengrass:StartBulkDeployment greengrass:StopBulkDeployment greengrass:UpdateConnectivityInfo greengrass:UpdateConnectorDefinition greengrass:UpdateCoreDefinition greengrass:UpdateDeviceDefinition greengrass:UpdateFunctionDefinition greengrass:UpdateGroup greengrass:UpdateGroupCertificateConfiguration greengrass:UpdateLoggerDefinition greengrass:UpdateResourceDefinition

Service prefix	Actions
	greengrass:UpdateSubscriptionDefinition greengrass:UpdateThingRuntimeConfiguration

Service prefix	Actions
groundstation	groundstation:CancelContact
	groundstation:CreateConfig
	groundstation:CreateDataflowEndpointGroup
	groundstation:CreateDataflowEndpointGroupV2
	groundstation:CreateEphemeris
	groundstation:CreateMissionProfile
	groundstation:DeleteConfig
	groundstation:DeleteDataflowEndpointGroup
	groundstation:DeleteEphemeris
	groundstation:DeleteMissionProfile
	groundstation:DescribeContact
	groundstation:DescribeContactVersion
	groundstation:DescribeEphemeris
	groundstation:GetConfig
	groundstation:GetDataflowEndpointGroup
	groundstation:GetMinuteUsage
	groundstation:GetMissionProfile
	groundstation:GetSatellite
	groundstation:ListAntennas
	groundstation:ListConfigs
	groundstation:ListContacts

Service prefix	Actions
	groundstation:ListDataflowEndpointGroups groundstation:ListEphemerides groundstation:ListGroundStationReservations groundstation:ListGroundStations groundstation:ListMissionProfiles groundstation:ListSatellites groundstation:RegisterAgent groundstation:ReserveContact groundstation:UpdateAgentStatus groundstation:UpdateConfig groundstation:UpdateEphemeris groundstation:UpdateMissionProfile

Service prefix	Actions
guardduty	guardduty:AcceptAdministratorInvitation guardduty:AcceptInvitation guardduty:ArchiveFindings guardduty:CreateDetector guardduty:CreateFilter guardduty:CreateIPSet guardduty:CreateInvestigation guardduty:CreateMalwareProtectionPlan guardduty:CreateMembers guardduty:CreatePublishingDestination guardduty:CreateSampleFindings guardduty:CreateThreatEntitySet guardduty:CreateThreatIntelSet guardduty:CreateTrustedEntitySet guardduty:DeclineInvitations guardduty:DeleteDetector guardduty:DeleteFilter guardduty:DeleteIPSet guardduty:DeleteInvitations guardduty:DeleteMalwareProtectionPlan guardduty:DeleteMembers

Service prefix	Actions
	guardduty:DeletePublishingDestination guardduty:DeleteThreatEntitySet guardduty:DeleteThreatIntelSet guardduty:DeleteTrustedEntitySet guardduty:DescribeMalwareScans guardduty:DescribeOrganizationConfiguration guardduty:DescribePublishingDestination guardduty:DisableOrganizationAdminAccount guardduty:DisassociateFromAdministratorAccount guardduty:DisassociateFromMasterAccount guardduty:DisassociateMembers guardduty:EnableOrganizationAdminAccount guardduty:GetAdministratorAccount guardduty:GetCoverageStatistics guardduty:GetDetector guardduty:GetFilter guardduty:GetFindings guardduty:GetFindingsStatistics guardduty:GetIPSet guardduty:GetInvestigation guardduty:GetInvitationsCount

Service prefix	Actions
	guardduty:GetMalwareProtectionPlan guardduty:GetMalwareScan guardduty:GetMalwareScanSettings guardduty:GetMasterAccount guardduty:GetMemberDetectors guardduty:GetMembers guardduty:GetOrganizationStatistics guardduty:GetRemainingFreeTrialDays guardduty:GetThreatEntitySet guardduty:GetThreatIntelSet guardduty:GetTrustedEntitySet guardduty:GetUsageStatistics guardduty:InviteMembers guardduty:ListCoverage guardduty:ListDetectors guardduty:ListFilters guardduty:ListFindings guardduty:ListIPSets guardduty:ListInvestigations guardduty:ListInvitations guardduty:ListMalwareProtectionPlans

Service prefix	Actions
	guardduty:ListMalwareScans guardduty:ListMembers guardduty:ListOrganizationAdminAccounts guardduty:ListPublishingDestinations guardduty:ListThreatEntitySets guardduty:ListThreatIntelSets guardduty:ListTrustedEntitySets guardduty:StartMalwareScan guardduty:StartMonitoringMembers guardduty:StopMonitoringMembers guardduty:UnarchiveFindings guardduty:UpdateDetector guardduty:UpdateFilter guardduty:UpdateFindingsFeedback guardduty:UpdateIPSet guardduty:UpdateMalwareProtectionPlan guardduty:UpdateMalwareScanSettings guardduty:UpdateMemberDetectors guardduty:UpdateOrganizationConfiguration guardduty:UpdatePublishingDestination guardduty:UpdateThreatEntitySet

Service prefix	Actions
	guardduty:UpdateThreatIntelSet guardduty:UpdateTrustedEntitySet

Service prefix	Actions
healthlake	healthlake:CancelFHIRExportJobWithDelete
	healthlake:CreateDataTransformationProfile
	healthlake:CreateFHIRDatastore
	healthlake:CreateResource
	healthlake:DeleteDataTransformationProfile
	healthlake:DeleteFHIRDatastore
	healthlake:DeleteResource
	healthlake:DescribeDataTransformationJob
	healthlake:DescribeFHIRDatastore
	healthlake:DescribeFHIRExportJob
	healthlake:DescribeFHIRExportJobWithGet
	healthlake:DescribeFHIRImportJob
	healthlake:GetCapabilities
	healthlake:GetDataTransformationProfile
	healthlake:ListDataTransformationJobs
	healthlake:ListDataTransformationProfileVersions
	healthlake:ListDataTransformationProfiles
	healthlake:ListFHIRDatastores
	healthlake:ListFHIRExportJobs
	healthlake:ListFHIRImportJobs
	healthlake:PublishDataTransformationProfile

Service prefix	Actions
	healthlake:ReadResource
	healthlake:SearchEverything
	healthlake:SearchWithGet
	healthlake:SearchWithPost
	healthlake:StartFHIRExportJob
	healthlake:StartFHIRExportJobWithPost
	healthlake:StartFHIRImportJob
	healthlake:UpdateDataTransformationProfile
	healthlake:UpdateFHIRDatastore
	healthlake:UpdateProfileWithAgent
	healthlake:UpdateResource

Service prefix	Actions
honeycode	honeycode:BatchCreateTableRows honeycode:BatchDeleteTableRows honeycode:BatchUpdateTableRows honeycode:BatchUpsertTableRows honeycode:DescribeTableDataImportJob honeycode:GetScreenData honeycode:InvokeScreenAutomation honeycode:ListTableColumns honeycode:ListTableRows honeycode:ListTables honeycode:QueryTableRows honeycode:StartTableDataImportJob

Service prefix	Actions
iam	iam:AddClientIDToOpenIDConnectProvider
	iam:AddRoleToInstanceProfile
	iam:AddUserToGroup
	iam:AttachGroupPolicy
	iam:AttachRolePolicy
	iam:AttachUserPolicy
	iam:ChangePassword
	iam:CreateAccessKey
	iam:CreateAccountAlias
	iam:CreateGroup
	iam:CreateInstanceProfile
	iam:CreateLoginProfile
	iam:CreateOpenIDConnectProvider
	iam:CreatePolicy
	iam:CreatePolicyVersion
	iam:CreateRole
	iam:CreateSAMLProvider
	iam:CreateServiceLinkedRole
	iam:CreateServiceSpecificCredential
	iam:CreateUser
	iam:CreateVirtualMFADevice

Service prefix	Actions
	iam:DeactivateMFADevice
	iam:DeleteAccessKey
	iam:DeleteAccountAlias
	iam:DeleteAccountPasswordPolicy
	iam:DeleteCloudFrontPublicKey
	iam:DeleteGroup
	iam:DeleteGroupPolicy
	iam:DeleteInstanceProfile
	iam:DeleteLoginProfile
	iam:DeleteOpenIDConnectProvider
	iam:DeletePolicy
	iam:DeletePolicyVersion
	iam:DeleteRole
	iam:DeleteRolePermissionsBoundary
	iam:DeleteRolePolicy
	iam:DeleteSAMLProvider
	iam:DeleteSSHPublicKey
	iam:DeleteServerCertificate
	iam:DeleteServiceLinkedRole
	iam:DeleteServiceSpecificCredential
	iam:DeleteSigningCertificate

Service prefix	Actions
	iam:DeleteUser
	iam:DeleteUserPermissionsBoundary
	iam:DeleteUserPolicy
	iam:DeleteVirtualMFADevice
	iam:DetachGroupPolicy
	iam:DetachRolePolicy
	iam:DetachUserPolicy
	iam:DisableOrganizationsRootCredentialsManagement
	iam:DisableOrganizationsRootSessions
	iam:DisableOutboundWebIdentityFederation
	iam:EnableMFADevice
	iam:EnableOrganizationsRootCredentialsManagement
	iam:EnableOrganizationsRootSessions
	iam:EnableOutboundWebIdentityFederation
	iam:GenerateCredentialReport
	iam:GenerateOrganizationsAccessReport
	iam:GenerateServiceLastAccessedDetails
	iam:GetAccessKeyLastUsed
	iam:GetAccountAuthorizationDetails
	iam:GetAccountEmailAddress
	iam:GetAccountName

Service prefix	Actions
	iam:GetAccountPasswordPolicy
	iam:GetAccountSummary
	iam:GetCloudFrontPublicKey
	iam:GetContextKeysForCustomPolicy
	iam:GetContextKeysForPrincipalPolicy
	iam:GetCredentialReport
	iam:GetGroup
	iam:GetGroupPolicy
	iam:GetInstanceProfile
	iam:GetLoginProfile
	iam:GetMFADevice
	iam:GetOpenIDConnectProvider
	iam:GetOrganizationsAccessReport
	iam:GetOutboundWebIdentityFederationInfo
	iam:GetPolicy
	iam:GetPolicyVersion
	iam:GetRole
	iam:GetRolePolicy
	iam:GetSAMLProvider
	iam:GetSSHPublicKey
	iam:GetServerCertificate

Service prefix	Actions
	iam:GetServiceLastAccessedDetails
	iam:GetServiceLastAccessedDetailsWithEntities
	iam:GetServiceLinkedRoleDeletionStatus
	iam:GetUser
	iam:GetUserPolicy
	iam:ListAccessKeys
	iam:ListAccountAliases
	iam:ListAttachedGroupPolicies
	iam:ListAttachedRolePolicies
	iam:ListAttachedUserPolicies
	iam:ListCloudFrontPublicKeys
	iam:ListDelegationRequests
	iam:ListEntitiesForPolicy
	iam:ListGroupPolicies
	iam:ListGroups
	iam:ListGroupsForUser
	iam:ListInstanceProfiles
	iam:ListInstanceProfilesForRole
	iam:ListMFADevices
	iam:ListOpenIDConnectProviders
	iam:ListOrganizationsFeatures

Service prefix	Actions
	iam:ListPolicies
	iam:ListPoliciesGrantingServiceAccess
	iam:ListPolicyVersions
	iam:ListRolePolicies
	iam:ListRoles
	iam:ListSAMLProviders
	iam:ListSSHPublicKeys
	iam:ListSTSRegionalEndpointsStatus
	iam:ListServerCertificates
	iam:ListServiceSpecificCredentials
	iam:ListSigningCertificates
	iam:ListUserPolicies
	iam:ListUsers
	iam:ListVirtualMFADevices
	iam:PutGroupPolicy
	iam:PutRolePermissionsBoundary
	iam:PutRolePolicy
	iam:PutUserPermissionsBoundary
	iam:PutUserPolicy
	iam:RemoveClientIDFromOpenIDConnectProvider
	iam:RemoveRoleFromInstanceProfile

Service prefix	Actions
	iam:RemoveUserFromGroup
	iam:ResetServiceSpecificCredential
	iam:ResyncMFADevice
	iam:SetDefaultPolicyVersion
	iam:SetSTSRegionalEndpointStatus
	iam:SetSecurityTokenServicePreferences
	iam:SimulateCustomPolicy
	iam:SimulatePrincipalPolicy
	iam:UpdateAccessKey
	iam:UpdateAccountEmailAddress
	iam:UpdateAccountName
	iam:UpdateAccountPasswordPolicy
	iam:UpdateAssumeRolePolicy
	iam:UpdateCloudFrontPublicKey
	iam:UpdateGroup
	iam:UpdateLoginProfile
	iam:UpdateOpenIDConnectProviderThumbprint
	iam:UpdateRole
	iam:UpdateRoleDescription
	iam:UpdateSAMLProvider
	iam:UpdateSSHPublicKey

Service prefix	Actions
	iam:UpdateServerCertificate iam:UpdateServiceSpecificCredential iam:UpdateSigningCertificate iam:UpdateUser iam:UploadCloudFrontPublicKey iam:UploadSSHPublicKey iam:UploadServerCertificate iam:UploadSigningCertificate

Service prefix	Actions
identitystore	identitystore:CreateGroup identitystore:CreateGroupMembership identitystore:CreateUser identitystore:DeleteGroup identitystore:DeleteGroupMembership identitystore:DeleteUser identitystore:DescribeGroup identitystore:DescribeGroupMembership identitystore:DescribeUser identitystore:GetGroupId identitystore:GetGroupMembershipId identitystore:GetUserId identitystore:IsMemberInGroups identitystore:ListGroupMemberships identitystore:ListGroupMembershipsForMember identitystore:ListGroups identitystore:ListUsers identitystore:UpdateGroup identitystore:UpdateUser

Service prefix	Actions
imagebuilder	imagebuilder:CancelImageCreation
	imagebuilder:CancelLifecycleExecution
	imagebuilder:CreateComponent
	imagebuilder:CreateContainerRecipe
	imagebuilder:CreateDistributionConfiguration
	imagebuilder:CreateImage
	imagebuilder:CreateImagePipeline
	imagebuilder:CreateImageRecipe
	imagebuilder:CreateInfrastructureConfiguration
	imagebuilder:CreateLifecyclePolicy
	imagebuilder:CreateWorkflow
	imagebuilder:DeleteComponent
	imagebuilder:DeleteContainerRecipe
	imagebuilder:DeleteDistributionConfiguration
	imagebuilder:DeleteImage
	imagebuilder:DeleteImagePipeline
	imagebuilder:DeleteImageRecipe
	imagebuilder:DeleteInfrastructureConfiguration
	imagebuilder:DeleteLifecyclePolicy
	imagebuilder:DeleteWorkflow
	imagebuilder:DistributeImage

Service prefix	Actions
	imagebuilder:GetComponentPolicy
	imagebuilder:GetContainerRecipePolicy
	imagebuilder:GetImagePolicy
	imagebuilder:GetImageRecipePolicy
	imagebuilder:GetLifecycleExecution
	imagebuilder:GetLifecyclePolicy
	imagebuilder:GetMarketplaceResource
	imagebuilder:GetWorkflowExecution
	imagebuilder:GetWorkflowStepExecution
	imagebuilder:ImportComponent
	imagebuilder:ImportDiskImage
	imagebuilder:ImportVmImage
	imagebuilder:ListComponentBuildVersions
	imagebuilder:ListComponents
	imagebuilder:ListContainerRecipes
	imagebuilder:ListDistributionConfigurations
	imagebuilder:ListImageBuildVersions
	imagebuilder:ListImagePackages
	imagebuilder:ListImagePipelineImages
	imagebuilder:ListImagePipelines
	imagebuilder:ListImageRecipes

Service prefix	Actions
	imagebuilder:ListImageScanFindingAggregations
	imagebuilder:ListImageScanFindings
	imagebuilder:ListImages
	imagebuilder:ListInfrastructureConfigurations
	imagebuilder:ListLifecycleExecutionResources
	imagebuilder:ListLifecycleExecutions
	imagebuilder:ListLifecyclePolicies
	imagebuilder:ListWaitingWorkflowSteps
	imagebuilder:ListWorkflowExecutions
	imagebuilder:ListWorkflowStepExecutions
	imagebuilder:ListWorkflows
	imagebuilder:PutComponentPolicy
	imagebuilder:PutContainerRecipePolicy
	imagebuilder:PutImagePolicy
	imagebuilder:PutImageRecipePolicy
	imagebuilder:RetryImage
	imagebuilder:SendWorkflowStepAction
	imagebuilder:StartImagePipelineExecution
	imagebuilder:StartResourceStateUpdate
	imagebuilder:UpdateDistributionConfiguration
	imagebuilder:UpdateImagePipeline

Service prefix	Actions
	imagebuilder:UpdateInfrastructureConfiguration

Service prefix	Actions
inspector	inspector:AddAttributesToFindings inspector:CreateAssessmentTarget inspector:CreateAssessmentTemplate inspector:CreateExclusionsPreview inspector:CreateResourceGroup inspector:DeleteAssessmentRun inspector:DeleteAssessmentTarget inspector:DeleteAssessmentTemplate inspector:DescribeAssessmentRuns inspector:DescribeAssessmentTargets inspector:DescribeAssessmentTemplates inspector:DescribeCrossAccountAccessRole inspector:DescribeExclusions inspector:DescribeFindings inspector:DescribeResourceGroups inspector:DescribeRulesPackages inspector:GetAssessmentReport inspector:GetExclusionsPreview inspector:GetTelemetryMetadata inspector:ListAssessmentRunAgents inspector:ListAssessmentRuns

Service prefix	Actions
	inspector:ListAssessmentTargets inspector:ListAssessmentTemplates inspector:ListEventSubscriptions inspector:ListExclusions inspector:ListFindings inspector:ListRulesPackages inspector:PreviewAgents inspector:RegisterCrossAccountAccessRole inspector:RemoveAttributesFromFindings inspector:StartAssessmentRun inspector:StopAssessmentRun inspector:SubscribeToEvent inspector:UnsubscribeFromEvent inspector:UpdateAssessmentTarget

Service prefix	Actions
inspector2	inspector2:AssociateMember
	inspector2:BatchGetAccountStatus
	inspector2:BatchGetCodeSnippet
	inspector2:BatchGetFindingDetails
	inspector2:BatchGetFreeTrialInfo
	inspector2:BatchGetMemberEc2DeepInspectionStatus
	inspector2:BatchUpdateMemberEc2DeepInspectionStatus
	inspector2:CancelFindingsReport
	inspector2:CancelSbomExport
	inspector2:CreateCisScanConfiguration
	inspector2:CreateCodeSecurityIntegration
	inspector2:CreateConnector
	inspector2:CreateFilter
	inspector2:CreateFindingsReport
	inspector2:CreateSbomExport
	inspector2:DeleteCisScanConfiguration
	inspector2:DeleteCodeSecurityIntegration
	inspector2:DeleteConnector
	inspector2:DeleteFilter
	inspector2:DescribeOrganizationConfiguration
	inspector2:Disable

Service prefix	Actions
	inspector2:DisableDelegatedAdminAccount
	inspector2:DisassociateMember
	inspector2:Enable
	inspector2:EnableDelegatedAdminAccount
	inspector2:GetCisScanReport
	inspector2:GetCisScanResultDetails
	inspector2:GetClustersForImage
	inspector2:GetCodeSecurityIntegration
	inspector2:GetCodeSecurityScan
	inspector2:GetConfiguration
	inspector2:GetDelegatedAdminAccount
	inspector2:GetEc2DeepInspectionConfiguration
	inspector2:GetEncryptionKey
	inspector2:GetFindingsReportStatus
	inspector2:GetMember
	inspector2:GetSbomExport
	inspector2:ListAccountPermissions
	inspector2:ListCisScanConfigurations
	inspector2:ListCisScanResultsAggregatedByChecks
	inspector2:ListCisScanResultsAggregatedByTargetResource
	inspector2:ListCisScans

Service prefix	Actions
	inspector2:ListCodeSecurityIntegrations
	inspector2:ListCodeSecurityScanConfigurations
	inspector2:ListConnectorScanConfigurations
	inspector2:ListConnectors
	inspector2:ListCoverage
	inspector2:ListCoverageStatistics
	inspector2:ListDelegatedAdminAccounts
	inspector2:ListFilters
	inspector2:ListFindingAggregations
	inspector2:ListFindings
	inspector2:ListMembers
	inspector2:ListUsageTotals
	inspector2:ResetEncryptionKey
	inspector2:SearchVulnerabilities
	inspector2:SendCisSessionHealth
	inspector2:SendCisSessionTelemetry
	inspector2:StartCisSession
	inspector2:StartCodeSecurityScan
	inspector2:StopCisSession
	inspector2:UpdateCisScanConfiguration
	inspector2:UpdateCodeSecurityIntegration

Service prefix	Actions
	inspector2:UpdateConfiguration inspector2:UpdateConnector inspector2:UpdateEc2DeepInspectionConfiguration inspector2:UpdateEncryptionKey inspector2:UpdateFilter inspector2:UpdateOrgEc2DeepInspectionConfiguration inspector2:UpdateOrganizationConfiguration

Service prefix	Actions
iot	iot:AcceptCertificateTransfer iot:AddThingToBillingGroup iot:AddThingToThingGroup iot:AssociateSbomWithPackageVersion iot:AssociateTargetsWithJob iot:AttachPolicy iot:AttachPrincipalPolicy iot:AttachSecurityProfile iot:AttachThingPrincipal iot:CancelAuditMitigationActionsTask iot:CancelAuditTask iot:CancelCertificateTransfer iot:CancelDetectMitigationActionsTask iot:CancelJob iot:CancelJobExecution iot:ClearDefaultAuthorizer iot:ConfirmTopicRuleDestination iot:CreateAuditSuppression iot:CreateAuthorizer iot:CreateBillingGroup iot:CreateCertificateFromCsr

Service prefix	Actions
	iot:CreateCertificateProvider iot:CreateCommand iot:CreateCustomMetric iot:CreateDimension iot:CreateDomainConfiguration iot:CreateDynamicThingGroup iot:CreateFleetMetric iot:CreateJob iot:CreateJobTemplate iot:CreateKeysAndCertificate iot:CreateMitigationAction iot:CreateOTAUpdate iot:CreatePackage iot:CreatePackageVersion iot:CreatePolicy iot:CreatePolicyVersion iot:CreateProvisioningClaim iot:CreateProvisioningTemplate iot:CreateProvisioningTemplateVersion iot:CreateRoleAlias iot:CreateScheduledAudit

Service prefix	Actions
	iot:CreateSecurityProfile iot:CreateStream iot:CreateThing iot:CreateThingGroup iot:CreateThingType iot:CreateTopicRule iot:CreateTopicRuleDestination iot:DeleteAccountAuditConfiguration iot:DeleteAuditSuppression iot:DeleteAuthorizer iot:DeleteBillingGroup iot:DeleteCACertificate iot:DeleteCertificate iot:DeleteCertificateProvider iot:DeleteCommand iot:DeleteCustomMetric iot:DeleteDimension iot:DeleteDomainConfiguration iot:DeleteDynamicThingGroup iot:DeleteFleetMetric iot:DeleteJob

Service prefix	Actions
	iot:DeleteJobExecution iot:DeleteJobTemplate iot:DeleteMitigationAction iot:DeleteOTAUpdate iot:DeletePackage iot:DeletePackageVersion iot:DeletePolicy iot:DeletePolicyVersion iot:DeleteProvisioningTemplate iot:DeleteProvisioningTemplateVersion iot:DeleteRegistrationCode iot:DeleteRoleAlias iot:DeleteScheduledAudit iot:DeleteSecurityProfile iot:DeleteStream iot:DeleteThing iot:DeleteThingGroup iot:DeleteThingType iot:DeleteTopicRule iot:DeleteTopicRuleDestination iot:DeleteV2LoggingLevel

Service prefix	Actions
	iot:DeprecateThingType iot:DescribeAccountAuditConfiguration iot:DescribeAuditFinding iot:DescribeAuditMitigationActionsTask iot:DescribeAuditSuppression iot:DescribeAuditTask iot:DescribeAuthorizer iot:DescribeBillingGroup iot:DescribeCACertificate iot:DescribeCertificate iot:DescribeCertificateProvider iot:DescribeCustomMetric iot:DescribeDefaultAuthorizer iot:DescribeDetectMitigationActionsTask iot:DescribeDimension iot:DescribeDomainConfiguration iot:DescribeEncryptionConfiguration iot:DescribeEndpoint iot:DescribeEventConfigurations iot:DescribeFleetMetric iot:DescribeIndex

Service prefix	Actions
	iot:DescribeJob iot:DescribeJobExecution iot:DescribeJobTemplate iot:DescribeManagedJobTemplate iot:DescribeMitigationAction iot:DescribeProvisioningTemplate iot:DescribeProvisioningTemplateVersion iot:DescribeRoleAlias iot:DescribeScheduledAudit iot:DescribeSecurityProfile iot:DescribeStream iot:DescribeThing iot:DescribeThingGroup iot:DescribeThingRegistrationTask iot:DescribeThingType iot:DetachPolicy iot:DetachPrincipalPolicy iot:DetachSecurityProfile iot:DetachThingPrincipal iot:DisableTopicRule iot:DisassociateSbomFromPackageVersion

Service prefix	Actions
	iot:EnableTopicRule iot:GetBehaviorModelTrainingSummaries iot:GetBucketsAggregation iot:GetCardinality iot:GetCommand iot:GetEffectivePolicies iot:GetJobDocument iot:GetLoggingOptions iot:GetOTAUpdate iot:GetPackage iot:GetPackageConfiguration iot:GetPackageVersion iot:GetPercentiles iot:GetPolicy iot:GetPolicyVersion iot:GetRegistrationCode iot:GetStatistics iot:GetThingConnectivityData iot:GetTopicRule iot:GetTopicRuleDestination iot:GetV2LoggingOptions

Service prefix	Actions
	iot:ListActiveViolations iot:ListAttachedPolicies iot:ListAuditFindings iot:ListAuditMitigationActionsExecutions iot:ListAuditMitigationActionsTasks iot:ListAuditSuppressions iot:ListAuditTasks iot:ListAuthorizers iot:ListBillingGroups iot:ListCACertificates iot:ListCertificateProviders iot:ListCertificates iot:ListCertificatesByCA iot:ListCommands iot:ListCustomMetrics iot:ListDetectMitigationActionsExecutions iot:ListDetectMitigationActionsTasks iot:ListDimensions iot:ListDomainConfigurations iot:ListFleetMetrics iot:ListIndices

Service prefix	Actions
	iot:ListJobExecutionsForJob iot:ListJobExecutionsForThing iot:ListJobTemplates iot:ListJobs iot:ListManagedJobTemplates iot:ListMetricValues iot:ListMitigationActions iot:ListOTAUpdates iot:ListOutgoingCertificates iot:ListPackageVersions iot:ListPackages iot:ListPolicies iot:ListPolicyPrincipals iot:ListPolicyVersions iot:ListPrincipalPolicies iot:ListPrincipalThings iot:ListPrincipalThingsV2 iot:ListProvisioningTemplateVersions iot:ListProvisioningTemplates iot:ListRelatedResourcesForAuditFinding iot:ListRoleAliases

Service prefix	Actions
	iot:ListSbomValidationResults iot:ListScheduledAudits iot:ListSecurityProfiles iot:ListSecurityProfilesForTarget iot:ListStreams iot:ListTargetsForPolicy iot:ListTargetsForSecurityProfile iot:ListThingGroups iot:ListThingGroupsForThing iot:ListThingPrincipals iot:ListThingPrincipalsV2 iot:ListThingRegistrationTaskReports iot:ListThingRegistrationTasks iot:ListThingTypes iot:ListThings iot:ListThingsInBillingGroup iot:ListThingsInThingGroup iot:ListTopicRuleDestinations iot:ListTopicRules iot:ListV2LoggingLevels iot:ListViolationEvents

Service prefix	Actions
	iot:PutVerificationStateOnViolation iot:RegisterCACertificate iot:RegisterCertificate iot:RegisterCertificateWithoutCA iot:RegisterThing iot:RejectCertificateTransfer iot:RemoveThingFromBillingGroup iot:RemoveThingFromThingGroup iot:ReplaceTopicRule iot:SearchIndex iot:SetDefaultAuthorizer iot:SetDefaultPolicyVersion iot:SetLoggingOptions iot:SetV2LoggingLevel iot:SetV2LoggingOptions iot:StartAuditMitigationActionsTask iot:StartDetectMitigationActionsTask iot:StartOnDemandAuditTask iot:StartThingRegistrationTask iot:StopThingRegistrationTask iot:TestAuthorization

Service prefix	Actions
	iot:TestInvokeAuthorizer iot:TransferCertificate iot:UpdateAccountAuditConfiguration iot:UpdateAuditSuppression iot:UpdateAuthorizer iot:UpdateBillingGroup iot:UpdateCACertificate iot:UpdateCertificate iot:UpdateCertificateProvider iot:UpdateCommand iot:UpdateCustomMetric iot:UpdateDimension iot:UpdateDomainConfiguration iot:UpdateDynamicThingGroup iot:UpdateEncryptionConfiguration iot:UpdateEventConfigurations iot:UpdateFleetMetric iot:UpdateIndexingConfiguration iot:UpdateJob iot:UpdateMitigationAction iot:UpdatePackage

Service prefix	Actions
	iot:UpdatePackageConfiguration iot:UpdatePackageVersion iot:UpdateProvisioningTemplate iot:UpdateRoleAlias iot:UpdateScheduledAudit iot:UpdateSecurityProfile iot:UpdateStream iot:UpdateThing iot:UpdateThingGroup iot:UpdateThingGroupsForThing iot:UpdateThingType iot:UpdateTopicRuleDestination iot:ValidateSecurityProfileBehaviors

Service prefix	Actions
iotanalytics	iotanalytics:CancelPipelineReprocessing
	iotanalytics:CreateChannel
	iotanalytics:CreateDataset
	iotanalytics:CreateDatasetContent
	iotanalytics:CreateDatastore
	iotanalytics:CreatePipeline
	iotanalytics:DeleteChannel
	iotanalytics:DeleteDataset
	iotanalytics:DeleteDatasetContent
	iotanalytics:DeleteDatastore
	iotanalytics:DeletePipeline
	iotanalytics:DescribeChannel
	iotanalytics:DescribeDataset
	iotanalytics:DescribeDatastore
	iotanalytics:DescribeLoggingOptions
	iotanalytics:DescribePipeline
	iotanalytics:GetDatasetContent
	iotanalytics:ListChannels
	iotanalytics:ListDatasetContents
	iotanalytics:ListDatasets
	iotanalytics:ListDatastores

Service prefix	Actions
	iotanalytics:ListPipelines iotanalytics:PutLoggingOptions iotanalytics:RunPipelineActivity iotanalytics:SampleChannelData iotanalytics:StartPipelineReprocessing iotanalytics:UpdateChannel iotanalytics:UpdateDataset iotanalytics:UpdateDatastore iotanalytics:UpdatePipeline
iotdeviceadvisor	iotdeviceadvisor:CreateSuiteDefinition iotdeviceadvisor:DeleteSuiteDefinition iotdeviceadvisor:GetEndpoint iotdeviceadvisor:GetSuiteDefinition iotdeviceadvisor:GetSuiteRun iotdeviceadvisor:GetSuiteRunReport iotdeviceadvisor:ListSuiteDefinitions iotdeviceadvisor:ListSuiteRuns iotdeviceadvisor:StartSuiteRun iotdeviceadvisor:StopSuiteRun iotdeviceadvisor:UpdateSuiteDefinition

Service prefix	Actions
iotevents	iotevents:BatchAcknowledgeAlarm
	iotevents:BatchDeleteDetector
	iotevents:BatchDisableAlarm
	iotevents:BatchEnableAlarm
	iotevents:BatchResetAlarm
	iotevents:BatchSnoozeAlarm
	iotevents:BatchUpdateDetector
	iotevents:CreateAlarmModel
	iotevents:CreateDetectorModel
	iotevents:CreateInput
	iotevents:DeleteAlarmModel
	iotevents:DeleteDetectorModel
	iotevents:DeleteInput
	iotevents:DescribeAlarm
	iotevents:DescribeAlarmModel
	iotevents:DescribeDetector
	iotevents:DescribeDetectorModel
	iotevents:DescribeDetectorModelAnalysis
	iotevents:DescribeInput
	iotevents:DescribeLoggingOptions
	iotevents:GetDetectorModelAnalysisResults

Service prefix	Actions
	iotevents:ListAlarmModelVersions iotevents:ListAlarmModels iotevents:ListAlarms iotevents:ListDetectorModelVersions iotevents:ListDetectorModels iotevents:ListDetectors iotevents:ListInputRoutings iotevents:ListInputs iotevents:PutLoggingOptions iotevents:StartDetectorModelAnalysis iotevents:UpdateAlarmModel iotevents:UpdateDetectorModel iotevents:UpdateInput
iotfleethub	iotfleethub:CreateApplication iotfleethub:DeleteApplication iotfleethub:DescribeApplication iotfleethub:ListApplications iotfleethub:UpdateApplication

Service prefix	Actions
iotsitewise	iotsitewise:AssociateAssets
	iotsitewise:AssociateTimeSeriesToAssetProperty
	iotsitewise:BatchAssociateProjectAssets
	iotsitewise:BatchDisassociateProjectAssets
	iotsitewise:CreateAccessPolicy
	iotsitewise:CreateAsset
	iotsitewise:CreateAssetModel
	iotsitewise:CreateAssetModelCompositeModel
	iotsitewise:CreateBulkImportJob
	iotsitewise:CreateComputationModel
	iotsitewise:CreateDashboard
	iotsitewise:CreateDataset
	iotsitewise:CreateGateway
	iotsitewise:CreatePortal
	iotsitewise:CreateProject
	iotsitewise:DeleteAccessPolicy
	iotsitewise:DeleteAsset
	iotsitewise:DeleteAssetModel
	iotsitewise:DeleteAssetModelCompositeModel
	iotsitewise:DeleteComputationModel
	iotsitewise:DeleteDashboard

Service prefix	Actions
	iotsitewise:DeleteDataset
	iotsitewise:DeleteGateway
	iotsitewise:DeletePortal
	iotsitewise:DeleteProject
	iotsitewise:DeleteTimeSeries
	iotsitewise:DescribeAccessPolicy
	iotsitewise:DescribeAsset
	iotsitewise:DescribeAssetCompositeModel
	iotsitewise:DescribeAssetModel
	iotsitewise:DescribeAssetModelCompositeModel
	iotsitewise:DescribeAssetModelInterfaceRelationship
	iotsitewise:DescribeAssetProperty
	iotsitewise:DescribeBulkImportJob
	iotsitewise:DescribeComputationModel
	iotsitewise:DescribeComputationModelExecutionSummary
	iotsitewise:DescribeDashboard
	iotsitewise:DescribeDataset
	iotsitewise:DescribeDefaultEncryptionConfiguration
	iotsitewise:DescribeExecution
	iotsitewise:DescribeGateway
	iotsitewise:DescribeGatewayCapabilityConfiguration

Service prefix	Actions
	iotsitewise:DescribeLoggingOptions
	iotsitewise:DescribePortal
	iotsitewise:DescribeProject
	iotsitewise:DescribeStorageConfiguration
	iotsitewise:DescribeTimeSeries
	iotsitewise:DisassociateAssets
	iotsitewise:DisassociateTimeSeriesFromAssetProperty
	iotsitewise:ExecuteAction
	iotsitewise:ExecuteQuery
	iotsitewise:ListAccessPolicies
	iotsitewise:ListActions
	iotsitewise:ListAssetModelCompositeModels
	iotsitewise:ListAssetModelProperties
	iotsitewise:ListAssetModels
	iotsitewise:ListAssetProperties
	iotsitewise:ListAssetRelationships
	iotsitewise:ListAssets
	iotsitewise:ListAssociatedAssets
	iotsitewise:ListBulkImportJobs
	iotsitewise:ListCompositionRelationships
	iotsitewise:ListComputationModelDataBindingUsages

Service prefix	Actions
	iotsitewise:ListComputationModelResolveToResources
	iotsitewise:ListComputationModels
	iotsitewise:ListDashboards
	iotsitewise:ListDatasets
	iotsitewise:ListExecutions
	iotsitewise:ListGateways
	iotsitewise:ListInterfaceRelationships
	iotsitewise:ListPortals
	iotsitewise:ListProjectAssets
	iotsitewise:ListProjects
	iotsitewise:ListTimeSeries
	iotsitewise:PutDefaultEncryptionConfiguration
	iotsitewise:PutLoggingOptions
	iotsitewise:PutStorageConfiguration
	iotsitewise:UpdateAccessPolicy
	iotsitewise:UpdateAsset
	iotsitewise:UpdateAssetModel
	iotsitewise:UpdateAssetModelCompositeModel
	iotsitewise:UpdateAssetProperty
	iotsitewise:UpdateComputationModel
	iotsitewise:UpdateDashboard

Service prefix	Actions
	iotsitewise:UpdateDataset iotsitewise:UpdateGateway iotsitewise:UpdateGatewayCapabilityConfiguration iotsitewise:UpdatePortal iotsitewise:UpdateProject

Service prefix	Actions
iottwinmaker	iottwinmaker:CancelMetadataTransferJob iottwinmaker:CreateComponentType iottwinmaker:CreateEntity iottwinmaker:CreateMetadataTransferJob iottwinmaker:CreateScene iottwinmaker:CreateSyncJob iottwinmaker:CreateWorkspace iottwinmaker:DeleteComponentType iottwinmaker:DeleteEntity iottwinmaker:DeleteScene iottwinmaker:DeleteSyncJob iottwinmaker:DeleteWorkspace iottwinmaker:ExecuteQuery iottwinmaker:GetMetadataTransferJob iottwinmaker:GetPricingPlan iottwinmaker:GetScene iottwinmaker:GetSyncJob iottwinmaker:ListComponentTypes iottwinmaker:ListComponents iottwinmaker:ListEntities iottwinmaker:ListMetadataTransferJobs

Service prefix	Actions
	iottwinmaker:ListProperties iottwinmaker:ListScenes iottwinmaker:ListSyncJobs iottwinmaker:ListSyncResources iottwinmaker:ListWorkspaces iottwinmaker:UpdateComponentType iottwinmaker:UpdateEntity iottwinmaker:UpdatePricingPlan iottwinmaker:UpdateScene iottwinmaker:UpdateWorkspace

Service prefix	Actions
iotwireless	iotwireless:AssociateAwsAccountWithPartnerAccount
	iotwireless:AssociateMulticastGroupWithFuotaTask
	iotwireless:AssociateWirelessDeviceWithFuotaTask
	iotwireless:AssociateWirelessDeviceWithMulticastGroup
	iotwireless:AssociateWirelessDeviceWithThing
	iotwireless:AssociateWirelessGatewayWithCertificate
	iotwireless:AssociateWirelessGatewayWithThing
	iotwireless:CancelMulticastGroupSession
	iotwireless:CreateDestination
	iotwireless:CreateDeviceProfile
	iotwireless:CreateFuotaTask
	iotwireless:CreateMulticastGroup
	iotwireless:CreateNetworkAnalyzerConfiguration
	iotwireless:CreateServiceProfile
	iotwireless:CreateWirelessDevice
	iotwireless:CreateWirelessGateway
	iotwireless:CreateWirelessGatewayTask
	iotwireless:CreateWirelessGatewayTaskDefinition
	iotwireless:DeleteDestination
	iotwireless:DeleteDeviceProfile
	iotwireless:DeleteFuotaTask

Service prefix	Actions
	iotwireless:DeleteMulticastGroup
	iotwireless:DeleteNetworkAnalyzerConfiguration
	iotwireless:DeleteQueuedMessages
	iotwireless:DeleteServiceProfile
	iotwireless:DeleteWirelessDevice
	iotwireless:DeleteWirelessDeviceImportTask
	iotwireless:DeleteWirelessGateway
	iotwireless:DeleteWirelessGatewayTask
	iotwireless:DeleteWirelessGatewayTaskDefinition
	iotwireless:DeregisterWirelessDevice
	iotwireless:DisassociateAwsAccountFromPartnerAccount
	iotwireless:DisassociateMulticastGroupFromFuotaTask
	iotwireless:DisassociateWirelessDeviceFromFuotaTask
	iotwireless:DisassociateWirelessDeviceFromMulticastGroup
	iotwireless:DisassociateWirelessDeviceFromThing
	iotwireless:DisassociateWirelessGatewayFromCertificate
	iotwireless:DisassociateWirelessGatewayFromThing
	iotwireless:GetDestination
	iotwireless:GetDeviceProfile
	iotwireless:GetEventConfigurationByResourceTypes
	iotwireless:GetFuotaTask

Service prefix	Actions
	iotwireless:GetLogLevelsByResourceTypes
	iotwireless:GetMetricConfiguration
	iotwireless:GetMetrics
	iotwireless:GetMulticastGroup
	iotwireless:GetMulticastGroupSession
	iotwireless:GetNetworkAnalyzerConfiguration
	iotwireless:GetPartnerAccount
	iotwireless:GetPosition
	iotwireless:GetPositionConfiguration
	iotwireless:GetPositionEstimate
	iotwireless:GetResourceEventConfiguration
	iotwireless:GetResourceLogLevel
	iotwireless:GetResourcePosition
	iotwireless:GetServiceEndpoint
	iotwireless:GetServiceProfile
	iotwireless:GetWirelessDevice
	iotwireless:GetWirelessDeviceImportTask
	iotwireless:GetWirelessDeviceStatistics
	iotwireless:GetWirelessGateway
	iotwireless:GetWirelessGatewayCertificate
	iotwireless:GetWirelessGatewayFirmwareInformation

Service prefix	Actions
	iotwireless:GetWirelessGatewayStatistics
	iotwireless:GetWirelessGatewayTask
	iotwireless:GetWirelessGatewayTaskDefinition
	iotwireless:ListDestinations
	iotwireless:ListDeviceProfiles
	iotwireless:ListDevicesForWirelessDeviceImportTask
	iotwireless:ListEventConfigurations
	iotwireless:ListFuotaTasks
	iotwireless:ListMulticastGroups
	iotwireless:ListMulticastGroupsByFuotaTask
	iotwireless:ListNetworkAnalyzerConfigurations
	iotwireless:ListPartnerAccounts
	iotwireless:ListPositionConfigurations
	iotwireless:ListQueuedMessages
	iotwireless:ListServiceProfiles
	iotwireless:ListWirelessDeviceImportTasks
	iotwireless:ListWirelessDevices
	iotwireless:ListWirelessGatewayTaskDefinitions
	iotwireless:ListWirelessGateways
	iotwireless:PutPositionConfiguration
	iotwireless:PutResourceLogLevel

Service prefix	Actions
	iotwireless:ResetAllResourceLogLevels
	iotwireless:ResetResourceLogLevel
	iotwireless:SendDataToMulticastGroup
	iotwireless:SendDataToWirelessDevice
	iotwireless:StartBulkAssociateWirelessDeviceWithMulticastGroup
	iotwireless:StartBulkDisassociateWirelessDeviceFromMulticastGroup
	iotwireless:StartFuotaTask
	iotwireless:StartMulticastGroupSession
	iotwireless:StartNetworkAnalyzerStream
	iotwireless:StartSingleWirelessDeviceImportTask
	iotwireless:StartWirelessDeviceImportTask
	iotwireless:TestWirelessDevice
	iotwireless:UpdateDestination
	iotwireless:UpdateEventConfigurationByResourceTypes
	iotwireless:UpdateFuotaTask
	iotwireless:UpdateLogLevelsByResourceTypes
	iotwireless:UpdateMetricConfiguration
	iotwireless:UpdateMulticastGroup
	iotwireless:UpdateNetworkAnalyzerConfiguration
	iotwireless:UpdatePartnerAccount

Service prefix	Actions
	iotwireless:UpdatePosition iotwireless:UpdateResourceEventConfiguration iotwireless:UpdateResourcePosition iotwireless:UpdateWirelessDevice iotwireless:UpdateWirelessDeviceImportTask iotwireless:UpdateWirelessGateway

Service prefix	Actions
ivs	ivs:BatchGetChannel ivs:BatchGetStreamKey ivs:BatchStartViewerSessionRevocation ivs:CreateAdConfiguration ivs:CreateChannel ivs:CreateEncoderConfiguration ivs:CreateIngestConfiguration ivs:CreateParticipantToken ivs:CreatePlaybackRestrictionPolicy ivs:CreateRecordingConfiguration ivs:CreateStorageConfiguration ivs:CreateStreamKey ivs:DeleteAdConfiguration ivs:DeleteChannel ivs:DeleteEncoderConfiguration ivs:DeleteIngestConfiguration ivs:DeletePlaybackKeyPair ivs:DeletePlaybackRestrictionPolicy ivs:DeletePublicKey ivs:DeleteRecordingConfiguration ivs:DeleteStorageConfiguration

Service prefix	Actions
	ivs:DeleteStreamKey ivs:DisconnectParticipant ivs:GetAdConfiguration ivs:GetChannel ivs:GetComposition ivs:GetEncoderConfiguration ivs:GetIngestConfiguration ivs:GetParticipant ivs:GetPlaybackKeyPair ivs:GetPlaybackRestrictionPolicy ivs:GetPublicKey ivs:GetRecordingConfiguration ivs:GetStorageConfiguration ivs:GetStream ivs:GetStreamKey ivs:GetStreamSession ivs:ImportPlaybackKeyPair ivs:ImportPublicKey ivs:InsertAdBreak ivs:ListAdConfigurations ivs:ListChannels

Service prefix	Actions
	ivs:ListCompositions ivs:ListEncoderConfigurations ivs:ListIngestConfigurations ivs:ListParticipantEvents ivs:ListParticipantReplicas ivs:ListParticipants ivs:ListPlaybackKeyPairs ivs:ListPlaybackRestrictionPolicies ivs:ListPublicKeys ivs:ListRecordingConfigurations ivs:ListStorageConfigurations ivs:ListStreamKeys ivs:ListStreamSessions ivs:ListStreams ivs:PutMetadata ivs:StartComposition ivs:StartViewerSessionRevocation ivs:StopComposition ivs:StopStream ivs:UpdateAdConfiguration ivs:UpdateChannel

Service prefix	Actions
	ivs:UpdateIngestConfiguration ivs:UpdatePlaybackRestrictionPolicy
ivschat	ivschat:CreateChatToken ivschat:CreateLoggingConfiguration ivschat:CreateRoom ivschat:DeleteLoggingConfiguration ivschat:DeleteMessage ivschat:DeleteRoom ivschat:DisconnectUser ivschat:GetLoggingConfiguration ivschat:GetRoom ivschat:ListLoggingConfigurations ivschat:ListRooms ivschat:SendEvent ivschat:UpdateLoggingConfiguration ivschat:UpdateRoom

Service prefix	Actions
kafka	kafka:BatchAssociateScramSecret kafka:BatchDisassociateScramSecret kafka:CreateCluster kafka:CreateClusterV2 kafka:CreateConfiguration kafka:CreateReplicator kafka:CreateVpcConnection kafka:DeleteCluster kafka:DeleteClusterPolicy kafka:DeleteConfiguration kafka:DeleteReplicator kafka:DeleteVpcConnection kafka:DescribeCluster kafka:DescribeClusterOperation kafka:DescribeClusterOperationV2 kafka:DescribeClusterV2 kafka:DescribeConfiguration kafka:DescribeConfigurationRevision kafka:DescribeVpcConnection kafka:GetBootstrapBrokers kafka:GetClusterPolicy

Service prefix	Actions
	kafka:GetCompatibleKafkaVersions kafka:ListClientVpcConnections kafka:ListClusterOperations kafka:ListClusterOperationsV2 kafka:ListClusters kafka:ListClustersV2 kafka:ListConfigurationRevisions kafka:ListConfigurations kafka:ListKafkaVersions kafka:ListNodes kafka:ListReplicators kafka:ListScramSecrets kafka:ListVpcConnections kafka:PutClusterPolicy kafka:RebootBroker kafka:RejectClientVpcConnection kafka:UpdateBrokerCount kafka:UpdateBrokerStorage kafka:UpdateBrokerType kafka:UpdateClusterConfiguration kafka:UpdateClusterKafkaVersion

Service prefix	Actions
	kafka:UpdateConfiguration kafka:UpdateConnectivity kafka:UpdateMonitoring kafka:UpdateRebalancing kafka:UpdateReplicationInfo kafka:UpdateSecurity kafka:UpdateStorage
kafkaconnect	kafkaconnect:CreateConnector kafkaconnect:CreateCustomPlugin kafkaconnect:CreateWorkerConfiguration kafkaconnect:DeleteConnector kafkaconnect:DeleteCustomPlugin kafkaconnect:DeleteWorkerConfiguration kafkaconnect:DescribeConnector kafkaconnect:DescribeCustomPlugin kafkaconnect:DescribeWorkerConfiguration kafkaconnect:ListConnectorOperations kafkaconnect:ListConnectors kafkaconnect:ListCustomPlugins kafkaconnect:ListWorkerConfigurations kafkaconnect:UpdateConnector

Service prefix	Actions
kendra	kendra:AssociateEntitiesToExperience kendra:AssociatePersonasToEntities kendra:BatchDeleteDocument kendra:BatchDeleteFeaturedResultsSet kendra:BatchGetDocumentStatus kendra:BatchPutDocument kendra:ClearQuerySuggestions kendra:CreateAccessControlConfiguration kendra:CreateDataSource kendra:CreateExperience kendra:CreateFaq kendra:CreateFeaturedResultsSet kendra:CreateIndex kendra:CreateQuerySuggestionsBlockList kendra:CreateThesaurus kendra:DeleteDataSource kendra:DeleteExperience kendra:DeleteFaq kendra:DeleteIndex kendra:DeletePrincipalMapping kendra:DeleteQuerySuggestionsBlockList

Service prefix	Actions
	kendra:DeleteThesaurus kendra:DescribeAccessControlConfiguration kendra:DescribeDataSource kendra:DescribeExperience kendra:DescribeFaq kendra:DescribeFeaturedResultsSet kendra:DescribeIndex kendra:DescribePrincipalMapping kendra:DescribeQuerySuggestionsBlockList kendra:DescribeQuerySuggestionsConfig kendra:DescribeThesaurus kendra:DisassociateEntitiesFromExperience kendra:DisassociatePersonasFromEntities kendra:GetQuerySuggestions kendra:GetSnapshots kendra:ListAccessControlConfigurations kendra:ListDataSourceSyncJobs kendra:ListDataSources kendra:ListEntityPersonas kendra:ListExperienceEntities kendra:ListExperiences

Service prefix	Actions
	kendra:ListFaqs kendra:ListFeaturedResultsSets kendra:ListGroupsOlderThanOrderingId kendra:ListIndices kendra:ListQuerySuggestionsBlockLists kendra:ListThesauri kendra:PutPrincipalMapping kendra:Query kendra:Retrieve kendra:StartDataSourceSyncJob kendra:StopDataSourceSyncJob kendra:SubmitFeedback kendra:UpdateDataSource kendra:UpdateExperience kendra:UpdateFeaturedResultsSet kendra:UpdateIndex kendra:UpdateQuerySuggestionsBlockList kendra:UpdateQuerySuggestionsConfig kendra:UpdateThesaurus

Service prefix	Actions
kinesis	kinesis:CreateStream
	kinesis:DecreaseStreamRetentionPeriod
	kinesis:DeleteStream
	kinesis:DeregisterStreamConsumer
	kinesis:DescribeAccountSettings
	kinesis:DescribeLimits
	kinesis:DescribeStream
	kinesis:DescribeStreamConsumer
	kinesis:DescribeStreamSummary
	kinesis:DisableEnhancedMonitoring
	kinesis:EnableEnhancedMonitoring
	kinesis:IncreaseStreamRetentionPeriod
	kinesis:ListShards
	kinesis:ListStreamConsumers
	kinesis:ListStreams
	kinesis:MergeShards
	kinesis:RegisterStreamConsumer
	kinesis:SplitShard
	kinesis:StartStreamEncryption
	kinesis:StopStreamEncryption
	kinesis:UpdateAccountSettings

Service prefix	Actions
	kinesis:UpdateShardCount kinesis:UpdateStreamMode

Service prefix	Actions
kinesisanalytics	kinesisanalytics:AddApplicationCloudWatchLoggingOption kinesisanalytics:AddApplicationInput kinesisanalytics:AddApplicationInputProcessingConfiguration kinesisanalytics:AddApplicationOutput kinesisanalytics:AddApplicationReferenceDataSource kinesisanalytics:AddApplicationVpcConfiguration kinesisanalytics:CreateApplication kinesisanalytics:CreateApplicationPresignedUrl kinesisanalytics:CreateApplicationSnapshot kinesisanalytics:DeleteApplication kinesisanalytics:DeleteApplicationCloudWatchLoggingOption kinesisanalytics:DeleteApplicationInputProcessingConfiguration kinesisanalytics:DeleteApplicationOutput kinesisanalytics:DeleteApplicationReferenceDataSource kinesisanalytics:DeleteApplicationSnapshot kinesisanalytics:DeleteApplicationVpcConfiguration kinesisanalytics:DescribeApplication kinesisanalytics:DescribeApplicationOperation kinesisanalytics:DescribeApplicationSnapshot kinesisanalytics:DescribeApplicationVersion kinesisanalytics:DiscoverInputSchema

Service prefix	Actions
	kinesisanalytics:ListApplicationOperations kinesisanalytics:ListApplicationSnapshots kinesisanalytics:ListApplicationVersions kinesisanalytics:ListApplications kinesisanalytics:RollbackApplication kinesisanalytics:StartApplication kinesisanalytics:StopApplication kinesisanalytics:UpdateApplication kinesisanalytics:UpdateApplicationMaintenanceConfiguration

Service prefix	Actions
kms	kms:CancelKeyDeletion
	kms:ConnectCustomKeyStore
	kms:CreateAlias
	kms:CreateCustomKeyStore
	kms:CreateGrant
	kms:CreateKey
	kms:Decrypt
	kms:DeleteAlias
	kms:DeleteCustomKeyStore
	kms:DeleteImportedKeyMaterial
	kms:DeriveSharedSecret
	kms:DescribeCustomKeyStores
	kms:DescribeKey
	kms:DisableKey
	kms:DisableKeyRotation
	kms:DisconnectCustomKeyStore
	kms:EnableKey
	kms:EnableKeyRotation
	kms:Encrypt
	kms:GenerateDataKey
	kms:GenerateDataKeyPair

Service prefix	Actions
	kms:GenerateDataKeyPairWithoutPlaintext
	kms:GenerateDataKeyWithoutPlaintext
	kms:GenerateMac
	kms:GenerateRandom
	kms:GetKeyLastUsage
	kms:GetKeyPolicy
	kms:GetKeyRotationStatus
	kms:GetParametersForImport
	kms:GetPublicKey
	kms:ImportKeyMaterial
	kms:ListAliases
	kms:ListGrants
	kms:ListKeyPolicies
	kms:ListKeyRotations
	kms:ListKeys
	kms:ListRetirableGrants
	kms:ReplicateKey
	kms:RetireGrant
	kms:RevokeGrant
	kms:RotateKeyOnDemand
	kms:ScheduleKeyDeletion

Service prefix	Actions
	kms:Sign kms:UpdateAlias kms:UpdateCustomKeyStore kms:UpdateKeyDescription kms:UpdatePrimaryRegion kms:Verify kms:VerifyMac

Service prefix	Actions
lambda	lambda:AddLayerVersionPermission lambda:AddPermission lambda:CreateAlias lambda:CreateCodeSigningConfig lambda:CreateEventSourceMapping lambda:CreateFunction lambda:CreateFunctionUrlConfig lambda:DeleteAlias lambda:DeleteCapacityProvider lambda:DeleteCodeSigningConfig lambda:DeleteEventSourceMapping lambda:DeleteFunction lambda:DeleteFunctionCodeSigningConfig lambda:DeleteFunctionConcurrency lambda:DeleteFunctionEventInvokeConfig lambda:DeleteFunctionUrlConfig lambda:DeleteLayerVersion lambda:DeleteProvisionedConcurrencyConfig lambda:GetAccountSettings lambda:GetAlias lambda:GetCapacityProvider

Service prefix	Actions
	lambda:GetCodeSigningConfig lambda:GetEventSourceMapping lambda:GetFunction lambda:GetFunctionCodeSigningConfig lambda:GetFunctionConcurrency lambda:GetFunctionConfiguration lambda:GetFunctionEventInvokeConfig lambda:GetFunctionRecursionConfig lambda:GetFunctionScalingConfig lambda:GetFunctionUrlConfig lambda:GetLayerVersion lambda:GetLayerVersionPolicy lambda:GetPolicy lambda:GetProvisionedConcurrencyConfig lambda:GetRuntimeManagementConfig lambda:ListAliases lambda:ListCapacityProviders lambda:ListCodeSigningConfigs lambda:ListDurableExecutionsByFunction lambda:ListEventSourceMappings lambda:ListFunctionEventInvokeConfigs

Service prefix	Actions
	lambda:ListFunctionUrlConfigs lambda:ListFunctions lambda:ListFunctionsByCodeSigningConfig lambda:ListLayerVersions lambda:ListLayers lambda:ListProvisionedConcurrencyConfigs lambda:ListVersionsByFunction lambda:PublishLayerVersion lambda:PublishVersion lambda:PutFunctionCodeSigningConfig lambda:PutFunctionConcurrency lambda:PutFunctionEventInvokeConfig lambda:PutFunctionRecursionConfig lambda:PutFunctionScalingConfig lambda:PutProvisionedConcurrencyConfig lambda:PutRuntimeManagementConfig lambda:RemoveLayerVersionPermission lambda:RemovePermission lambda:UpdateAlias lambda:UpdateCapacityProvider lambda:UpdateCodeSigningConfig

Service prefix	Actions
	lambda:UpdateEventSourceMapping lambda:UpdateFunctionCode lambda:UpdateFunctionConfiguration lambda:UpdateFunctionEventInvokeConfig lambda:UpdateFunctionUrlConfig

Service prefix	Actions
lex	lex:BatchCreateCustomVocabularyItem
	lex:BatchDeleteCustomVocabularyItem
	lex:BatchUpdateCustomVocabularyItem
	lex:BuildBotLocale
	lex:CreateBotAlias
	lex:CreateBotReplica
	lex:CreateBotVersion
	lex:CreateExport
	lex:CreateIntentVersion
	lex:CreateResourcePolicy
	lex:CreateSlotTypeVersion
	lex:CreateTestSetDiscrepancyReport
	lex:CreateUploadUrl
	lex:DeleteBot
	lex:DeleteBotAnalyzerRecommendation
	lex:DeleteBotChannelAssociation
	lex:DeleteBotReplica
	lex:DeleteExport
	lex:DeleteImport
	lex:DeleteIntentVersion
	lex:DeleteResourcePolicy

Service prefix	Actions
	lex:DeleteSlotTypeVersion
	lex:DeleteTestSet
	lex:DeleteUtterances
	lex:DescribeBotAlias
	lex:DescribeBotAnalyzerRecommendation
	lex:DescribeBotRecommendation
	lex:DescribeBotReplica
	lex:DescribeBotResourceGeneration
	lex:DescribeBotVersion
	lex:DescribeCustomVocabularyMetadata
	lex:DescribeExport
	lex:DescribeImport
	lex:DescribeResourcePolicy
	lex:DescribeTestExecution
	lex:DescribeTestSet
	lex:DescribeTestSetDiscrepancyReport
	lex:DescribeTestSetGeneration
	lex:GenerateBotElement
	lex:GetBot
	lex:GetBotAlias
	lex:GetBotAliases

Service prefix	Actions
	lex:GetBotChannelAssociation
	lex:GetBotChannelAssociations
	lex:GetBotVersions
	lex:GetBots
	lex:GetBuiltinIntent
	lex:GetBuiltinIntents
	lex:GetBuiltinSlotTypes
	lex:GetExport
	lex:GetImport
	lex:GetIntent
	lex:GetIntentVersions
	lex:GetIntents
	lex:GetMigration
	lex:GetMigrations
	lex:GetSlotType
	lex:GetSlotTypeVersions
	lex:GetSlotTypes
	lex:GetTestExecutionArtifactsUrl
	lex:GetUtterancesView
	lex:ListBotAliasReplicas
	lex:ListBotAliases

Service prefix	Actions
	lex:ListBotRecommendations
	lex:ListBotReplicas
	lex:ListBotResourceGenerations
	lex:ListBotVersionReplicas
	lex:ListBotVersions
	lex:ListBots
	lex:ListBuiltInIntents
	lex:ListBuiltInSlotTypes
	lex:ListCustomVocabularyItems
	lex:ListExports
	lex:ListImports
	lex:ListIntentMetrics
	lex:ListIntentPaths
	lex:ListRecommendedIntents
	lex:ListSessionAnalyticsData
	lex:ListSessionMetrics
	lex:ListTestExecutionResultItems
	lex:ListTestExecutions
	lex:ListTestSets
	lex:PutBot
	lex:PutBotAlias

Service prefix	Actions
	lex:PutIntent
	lex:PutSlotType
	lex:SearchAssociatedTranscripts
	lex:StartBotAnalyzer
	lex:StartBotRecommendation
	lex:StartImport
	lex:StartMigration
	lex:StartTestExecution
	lex:StartTestSetGeneration
	lex:StopBotAnalyzer
	lex:StopBotRecommendation
	lex:UpdateBotAlias
	lex:UpdateBotRecommendation
	lex:UpdateExport
	lex:UpdateResourcePolicy

Service prefix	Actions
license-manager-linux-subscriptions	license-manager-linux-subscriptions:DeregisterSubscriptionProvider license-manager-linux-subscriptions:GetRegisteredSubscriptionProvider license-manager-linux-subscriptions:GetServiceSettings license-manager-linux-subscriptions:ListLinuxSubscriptionInstances license-manager-linux-subscriptions:ListLinuxSubscriptions license-manager-linux-subscriptions:ListRegisteredSubscriptionProviders license-manager-linux-subscriptions:RegisterSubscriptionProvider license-manager-linux-subscriptions:UpdateServiceSettings

Service prefix	Actions
lightsail	lightsail:AllocateStaticIp
	lightsail:AttachCertificateToDistribution
	lightsail:AttachDisk
	lightsail:AttachInstancesToLoadBalancer
	lightsail:AttachLoadBalancerTlsCertificate
	lightsail:AttachStaticIp
	lightsail:CloseInstancePublicPorts
	lightsail:CopySnapshot
	lightsail:CreateBucket
	lightsail:CreateBucketAccessKey
	lightsail:CreateCertificate
	lightsail:CreateCloudFormationStack
	lightsail:CreateContactMethod
	lightsail:CreateContainerService
	lightsail:CreateContainerServiceDeployment
	lightsail:CreateContainerServiceRegistryLogin
	lightsail:CreateDisk
	lightsail:CreateDiskFromSnapshot
	lightsail:CreateDiskSnapshot
	lightsail:CreateDistribution
	lightsail:CreateDomain

Service prefix	Actions
	lightsail:CreateGUISessionAccessDetails
	lightsail:CreateInstanceSnapshot
	lightsail:CreateInstances
	lightsail:CreateInstancesFromSnapshot
	lightsail:CreateKeyPair
	lightsail:CreateLoadBalancer
	lightsail:CreateLoadBalancerTlsCertificate
	lightsail:CreateRelationalDatabase
	lightsail:CreateRelationalDatabaseFromSnapshot
	lightsail:CreateRelationalDatabaseSnapshot
	lightsail:DeleteAlarm
	lightsail:DeleteAutoSnapshot
	lightsail:DeleteBucket
	lightsail:DeleteBucketAccessKey
	lightsail:DeleteCertificate
	lightsail:DeleteContactMethod
	lightsail:DeleteContainerImage
	lightsail:DeleteContainerService
	lightsail:DeleteDisk
	lightsail:DeleteDiskSnapshot
	lightsail:DeleteDistribution

Service prefix	Actions
	lightsail:DeleteDomain
	lightsail:DeleteDomainEntry
	lightsail:DeleteInstance
	lightsail:DeleteInstanceSnapshot
	lightsail:DeleteKeyPair
	lightsail:DeleteKnownHostKeys
	lightsail:DeleteLoadBalancer
	lightsail:DeleteLoadBalancerTlsCertificate
	lightsail:DeleteRelationalDatabase
	lightsail:DeleteRelationalDatabaseSnapshot
	lightsail:DetachCertificateFromDistribution
	lightsail:DetachDisk
	lightsail:DetachInstancesFromLoadBalancer
	lightsail:DetachStaticIp
	lightsail:DisableAddOn
	lightsail:DownloadDefaultKeyPair
	lightsail:EnableAddOn
	lightsail:ExportSnapshot
	lightsail:GetActiveNames
	lightsail:GetAlarms
	lightsail:GetAutoSnapshots

Service prefix	Actions
	lightsail:GetBlueprints
	lightsail:GetBucketAccessKeys
	lightsail:GetBucketBundles
	lightsail:GetBucketMetricData
	lightsail:GetBuckets
	lightsail:GetBundles
	lightsail:GetCertificates
	lightsail:GetCloudFormationStackRecords
	lightsail:GetContactMethods
	lightsail:GetContainerAPIMetadata
	lightsail:GetContainerImages
	lightsail:GetContainerLog
	lightsail:GetContainerServiceDeployments
	lightsail:GetContainerServiceMetricData
	lightsail:GetContainerServicePowers
	lightsail:GetContainerServices
	lightsail:GetCostEstimate
	lightsail:GetDisk
	lightsail:GetDiskSnapshot
	lightsail:GetDiskSnapshots
	lightsail:GetDisks

Service prefix	Actions
	lightsail:GetDistributionBundles
	lightsail:GetDistributionLatestCacheReset
	lightsail:GetDistributionMetricData
	lightsail:GetDistributions
	lightsail:GetDomain
	lightsail:GetExportSnapshotRecords
	lightsail:GetInstance
	lightsail:GetInstanceMetricData
	lightsail:GetInstancePortStates
	lightsail:GetInstanceSnapshot
	lightsail:GetInstanceSnapshots
	lightsail:GetInstanceState
	lightsail:GetInstances
	lightsail:GetKeyPair
	lightsail:GetKeyPairs
	lightsail:GetLoadBalancer
	lightsail:GetLoadBalancerMetricData
	lightsail:GetLoadBalancerTlsCertificates
	lightsail:GetLoadBalancerTlsPolicies
	lightsail:GetLoadBalancers
	lightsail:GetOperation

Service prefix	Actions
	lightsail:GetOperations
	lightsail:GetOperationsForResource
	lightsail:GetRegions
	lightsail:GetRelationalDatabase
	lightsail:GetRelationalDatabaseBlueprints
	lightsail:GetRelationalDatabaseBundles
	lightsail:GetRelationalDatabaseEvents
	lightsail:GetRelationalDatabaseLogEvents
	lightsail:GetRelationalDatabaseLogStreams
	lightsail:GetRelationalDatabaseMasterUserPassword
	lightsail:GetRelationalDatabaseMetricData
	lightsail:GetRelationalDatabaseParameters
	lightsail:GetRelationalDatabaseSnapshot
	lightsail:GetRelationalDatabaseSnapshots
	lightsail:GetRelationalDatabases
	lightsail:GetSetupHistory
	lightsail:GetStaticIp
	lightsail:GetStaticIps
	lightsail:ImportKeyPair
	lightsail:IsVpcPeered
	lightsail:OpenInstancePublicPorts

Service prefix	Actions
	lightsail:PeerVpc
	lightsail:PutAlarm
	lightsail:PutInstancePublicPorts
	lightsail:RebootInstance
	lightsail:RebootRelationalDatabase
	lightsail:RegisterContainerImage
	lightsail:ReleaseStaticIp
	lightsail:ResetDistributionCache
	lightsail:SendContactMethodVerification
	lightsail:SetIpAddressType
	lightsail:SetResourceAccessForBucket
	lightsail:SetupInstanceHttps
	lightsail:StartGUISession
	lightsail:StartInstance
	lightsail:StartRelationalDatabase
	lightsail:StopGUISession
	lightsail:StopInstance
	lightsail:StopRelationalDatabase
	lightsail:TestAlarm
	lightsail:UnpeerVpc
	lightsail:UpdateBucket

Service prefix	Actions
	lightsail:UpdateBucketBundle lightsail:UpdateContainerService lightsail:UpdateDistribution lightsail:UpdateDistributionBundle lightsail:UpdateDomainEntry lightsail:UpdateInstanceMetadataOptions lightsail:UpdateLoadBalancerAttribute lightsail:UpdateRelationalDatabase lightsail:UpdateRelationalDatabaseParameters

Service prefix	Actions
logs	logs:AssociateKmsKey logs:AssociateSourceToS3TableIntegration logs:CancelExportTask logs:CancelImportTask logs:CreateDelivery logs:CreateExportTask logs:CreateLogAnomalyDetector logs:CreateLogGroup logs:CreateLogStream logs:CreateLookupTable logs:DeleteDataProtectionPolicy logs:DeleteDelivery logs:DeleteDeliveryDestination logs:DeleteDeliveryDestinationPolicy logs:DeleteDeliverySource logs:DeleteDestination logs:DeleteIndexPolicy logs:DeleteIntegration logs:DeleteLogAnomalyDetector logs:DeleteLogGroup logs:DeleteLogStream

Service prefix	Actions
	logs:DeleteLookupTable logs:DeleteMetricFilter logs:DeleteQueryDefinition logs:DeleteResourcePolicy logs:DeleteRetentionPolicy logs:DeleteScheduledQuery logs:DeleteSubscriptionFilter logs:DeleteSyslogConfiguration logs:DeleteTransformer logs:DescribeAccountPolicies logs:DescribeConfigurationTemplates logs:DescribeDeliveries logs:DescribeDeliveryDestinations logs:DescribeDeliverySources logs:DescribeDestinations logs:DescribeExportTasks logs:DescribeFieldIndexes logs:DescribeImportTaskBatches logs:DescribeImportTasks logs:DescribeIndexPolicies logs:DescribeLogGroups

Service prefix	Actions
	logs:DescribeLogStreams logs:DescribeLookupTables logs:DescribeMetricFilters logs:DescribeQueries logs:DescribeQueryDefinitions logs:DescribeResourcePolicies logs:DescribeSubscriptionFilters logs:DisassociateKmsKey logs:DisassociateSourceFromS3TableIntegration logs:GetDataProtectionPolicy logs:GetDelivery logs:GetDeliveryDestination logs:GetDeliveryDestinationPolicy logs:GetDeliverySource logs:GetIntegration logs:GetLogAnomalyDetector logs:GetLogFields logs:GetLogGroupFields logs:GetLogRecord logs:GetLookupTable logs:GetQueryResults

Service prefix	Actions
	logs:GetScheduledQuery logs:GetScheduledQueryHistory logs:GetStorageTierPolicy logs:GetTransformer logs:ListAnomalies logs:ListIntegrations logs:ListLogAnomalyDetectors logs:ListLogGroupsForQuery logs:ListScheduledQueries logs:ListSourcesForS3TableIntegration logs:ListSyslogConfigurations logs:PutBearerTokenAuthentication logs:PutDataProtectionPolicy logs:PutDeliveryDestination logs:PutDeliveryDestinationPolicy logs:PutDeliverySource logs:PutDestination logs:PutDestinationPolicy logs:PutIndexPolicy logs:PutIntegration logs:PutLogGroupDeletionProtection

Service prefix	Actions
	logs:PutMetricFilter logs:PutQueryDefinition logs:PutResourcePolicy logs:PutRetentionPolicy logs:PutStorageTierPolicy logs:PutSubscriptionFilter logs:PutSyslogConfiguration logs:PutTransformer logs:StartLiveTail logs:StartQuery logs:StopQuery logs:TestMetricFilter logs:TestTransformer logs:UpdateAnomaly logs:UpdateDeliveryConfiguration logs:UpdateLogAnomalyDetector logs:UpdateLookupTable

Service prefix	Actions
lookoutequipment	lookoutequipment:CreateDataset
	lookoutequipment:CreateInferenceScheduler
	lookoutequipment:CreateLabel
	lookoutequipment:CreateLabelGroup
	lookoutequipment:CreateModel
	lookoutequipment:DeleteDataset
	lookoutequipment:DeleteInferenceScheduler
	lookoutequipment:DeleteLabel
	lookoutequipment:DeleteLabelGroup
	lookoutequipment:DeleteModel
	lookoutequipment:DeleteResourcePolicy
	lookoutequipment:DeleteRetrainingScheduler
	lookoutequipment:DescribeDataIngestionJob
	lookoutequipment:DescribeDataset
	lookoutequipment:DescribeInferenceScheduler
	lookoutequipment:DescribeLabelGroup
	lookoutequipment:DescribeModel
	lookoutequipment:DescribeModelVersion
	lookoutequipment:DescribeResourcePolicy
	lookoutequipment:DescribeRetrainingScheduler
	lookoutequipment:Describelabel

Service prefix	Actions
	lookoutequipment:ImportDataset
	lookoutequipment:ImportModelVersion
	lookoutequipment:ListDataIngestionJobs
	lookoutequipment:ListDatasets
	lookoutequipment:ListInferenceEvents
	lookoutequipment:ListInferenceExecutions
	lookoutequipment:ListInferenceSchedulers
	lookoutequipment:ListLabelGroups
	lookoutequipment:ListLabels
	lookoutequipment:ListModelVersions
	lookoutequipment:ListModels
	lookoutequipment:ListRetrainingSchedulers
	lookoutequipment:ListSensorStatistics
	lookoutequipment:PutResourcePolicy
	lookoutequipment:StartDataIngestionJob
	lookoutequipment:StartInferenceScheduler
	lookoutequipment:StartRetrainingScheduler
	lookoutequipment:StopInferenceScheduler
	lookoutequipment:StopRetrainingScheduler
	lookoutequipment:UpdateActiveModelVersion
	lookoutequipment:UpdateInferenceScheduler

Service prefix	Actions
	lookoutequipment:UpdateLabelGroup lookoutequipment:UpdateModel lookoutequipment:UpdateRetrainingScheduler

Service prefix	Actions
lookoutmetrics	lookoutmetrics:ActivateAnomalyDetector lookoutmetrics:BackTestAnomalyDetector lookoutmetrics:CreateAlert lookoutmetrics:CreateAnomalyDetector lookoutmetrics:CreateMetricSet lookoutmetrics:DeactivateAnomalyDetector lookoutmetrics:DeleteAlert lookoutmetrics:DeleteAnomalyDetector lookoutmetrics:DescribeAlert lookoutmetrics:DescribeAnomalyDetectionExecutions lookoutmetrics:DescribeAnomalyDetector lookoutmetrics:DescribeMetricSet lookoutmetrics:DetectMetricSetConfig lookoutmetrics:GetAnomalyGroup lookoutmetrics:GetDataQualityMetrics lookoutmetrics:GetFeedback lookoutmetrics:GetSampleData lookoutmetrics:ListAlerts lookoutmetrics:ListAnomalyDetectors lookoutmetrics:ListAnomalyGroupRelatedMetrics lookoutmetrics:ListAnomalyGroupSummaries

Service prefix	Actions
	lookoutmetrics:ListAnomalyGroupTimeSeries lookoutmetrics:ListMetricSets lookoutmetrics:PutFeedback lookoutmetrics:UpdateAlert lookoutmetrics:UpdateAnomalyDetector lookoutmetrics:UpdateMetricSet

Service prefix	Actions
m2	m2:CancelBatchJobExecution
	m2:CreateApplication
	m2:CreateDataSetExportTask
	m2:CreateDataSetImportTask
	m2:CreateDeployment
	m2:CreateEnvironment
	m2:DeleteApplication
	m2:DeleteApplicationFromEnvironment
	m2:DeleteEnvironment
	m2:GetApplication
	m2:GetApplicationVersion
	m2:GetBatchJobExecution
	m2:GetDataSetDetails
	m2:GetDataSetExportTask
	m2:GetDataSetImportTask
	m2:GetDeployment
	m2:GetEnvironment
	m2:GetSignedBluinsightsUrl
	m2:ListApplicationVersions
	m2:ListApplications
	m2:ListBatchJobDefinitions

Service prefix	Actions
	m2:ListBatchJobExecutions
	m2:ListBatchJobRestartPoints
	m2:ListDataSetExportHistory
	m2:ListDataSetImportHistory
	m2:ListDataSets
	m2:ListDeployments
	m2:ListEngineVersions
	m2:ListEnvironments
	m2:StartApplication
	m2:StartBatchJob
	m2:StopApplication
	m2:UpdateApplication
	m2:UpdateEnvironment

Service prefix	Actions
managedblockchain	managedblockchain:CreateAccessor managedblockchain:CreateMember managedblockchain:CreateNetwork managedblockchain:CreateNode managedblockchain:CreateProposal managedblockchain:DeleteAccessor managedblockchain:DeleteMember managedblockchain:DeleteNode managedblockchain:GetAccessor managedblockchain:GetMember managedblockchain:GetNetwork managedblockchain:GetNode managedblockchain:GetProposal managedblockchain:InvokeRpcPolygonMainnet managedblockchain:InvokeRpcPolygonMumbaiTestnet managedblockchain:ListAccessors managedblockchain:ListInvitations managedblockchain:ListMembers managedblockchain:ListNetworks managedblockchain:ListNodes managedblockchain:ListProposalVotes

Service prefix	Actions
	managedblockchain:ListProposals managedblockchain:RejectInvitation managedblockchain:UpdateMember managedblockchain:UpdateNode managedblockchain:VoteOnProposal

Service prefix	Actions
mediaconnect	mediaconnect:AddBridgeOutputs
	mediaconnect:AddBridgeSources
	mediaconnect:AddFlowMediaStreams
	mediaconnect:AddFlowOutputs
	mediaconnect:AddFlowSources
	mediaconnect:AddFlowVpcInterfaces
	mediaconnect:CreateBridge
	mediaconnect:CreateFlow
	mediaconnect:CreateGateway
	mediaconnect:DeleteBridge
	mediaconnect:DeleteFlow
	mediaconnect:DeleteGateway
	mediaconnect:DeleteRouterInput
	mediaconnect:DeleteRouterNetworkInterface
	mediaconnect:DeleteRouterOutput
	mediaconnect:DeregisterGatewayInstance
	mediaconnect:DescribeBridge
	mediaconnect:DescribeFlow
	mediaconnect:DescribeFlowSourceMetadata
	mediaconnect:DescribeFlowSourceThumbnail
	mediaconnect:DescribeGateway

Service prefix	Actions
	mediaconnect:DescribeGatewayInstance
	mediaconnect:DescribeOffering
	mediaconnect:DescribeReservation
	mediaconnect:GetRouterInput
	mediaconnect:GetRouterInputSourceMetadata
	mediaconnect:GetRouterInputThumbnail
	mediaconnect:GetRouterNetworkInterface
	mediaconnect:GetRouterOutput
	mediaconnect:GrantFlowEntitlements
	mediaconnect:ListBridges
	mediaconnect:ListEntitlements
	mediaconnect:ListFlows
	mediaconnect:ListGatewayInstances
	mediaconnect:ListGateways
	mediaconnect:ListOfferings
	mediaconnect:ListReservations
	mediaconnect:ListRouterInputs
	mediaconnect:ListRouterNetworkInterfaces
	mediaconnect:ListRouterOutputs
	mediaconnect:PurchaseOffering
	mediaconnect:RemoveBridgeOutput

Service prefix	Actions
	mediaconnect:RemoveBridgeSource
	mediaconnect:RemoveFlowMediaStream
	mediaconnect:RemoveFlowOutput
	mediaconnect:RemoveFlowSource
	mediaconnect:RemoveFlowVpcInterface
	mediaconnect:RestartRouterInput
	mediaconnect:RestartRouterOutput
	mediaconnect:RevokeFlowEntitlement
	mediaconnect:StartFlow
	mediaconnect:StartRouterInput
	mediaconnect:StartRouterOutput
	mediaconnect:StopFlow
	mediaconnect:StopRouterInput
	mediaconnect:StopRouterOutput
	mediaconnect:TakeRouterInput
	mediaconnect:UpdateBridge
	mediaconnect:UpdateBridgeOutput
	mediaconnect:UpdateBridgeSource
	mediaconnect:UpdateBridgeState
	mediaconnect:UpdateFlow
	mediaconnect:UpdateFlowEntitlement

Service prefix	Actions
	mediaconnect:UpdateFlowMediaStream mediaconnect:UpdateGatewayInstance

Service prefix	Actions
mediaconvert	mediaconvert:AssociateCertificate mediaconvert:CancelJob mediaconvert:CreateJob mediaconvert:CreateJobTemplate mediaconvert:CreatePreset mediaconvert:CreateQueue mediaconvert:CreateResourceShare mediaconvert:DeleteJobTemplate mediaconvert:DeletePolicy mediaconvert:DeletePreset mediaconvert:DeleteQueue mediaconvert:DescribeEndpoints mediaconvert:DisassociateCertificate mediaconvert:GetJob mediaconvert:GetJobTemplate mediaconvert:GetPolicy mediaconvert:GetPreset mediaconvert:GetQueue mediaconvert:ListJobTemplates mediaconvert:ListJobs mediaconvert:ListPresets

Service prefix	Actions
	mediaconvert:ListQueues mediaconvert:ListVersions mediaconvert:Probe mediaconvert:PutPolicy mediaconvert:SearchJobs mediaconvert:UpdateJobTemplate mediaconvert:UpdatePreset mediaconvert:UpdateQueue

Service prefix	Actions
medialive	medialive:AcceptInputDeviceTransfer
	medialive:BatchDelete
	medialive:BatchStart
	medialive:BatchStop
	medialive:BatchUpdateSchedule
	medialive:CancelInputDeviceTransfer
	medialive:ClaimDevice
	medialive:CreateChannel
	medialive:CreateChannelPlacementGroup
	medialive:CreateCloudWatchAlarmTemplate
	medialive:CreateCloudWatchAlarmTemplateGroup
	medialive:CreateCluster
	medialive:CreateEventBridgeRuleTemplate
	medialive:CreateEventBridgeRuleTemplateGroup
	medialive:CreateInput
	medialive:CreateInputSecurityGroup
	medialive:CreateMultiplex
	medialive:CreateMultiplexProgram
	medialive:CreateNetwork
	medialive:CreateNode
	medialive:CreateNodeRegistrationScript

Service prefix	Actions
	medialive:CreatePartnerInput
	medialive:CreateSdiSource
	medialive:CreateSignalMap
	medialive:DeleteChannel
	medialive:DeleteChannelPlacementGroup
	medialive:DeleteCloudWatchAlarmTemplate
	medialive:DeleteCloudWatchAlarmTemplateGroup
	medialive:DeleteCluster
	medialive:DeleteEventBridgeRuleTemplate
	medialive:DeleteEventBridgeRuleTemplateGroup
	medialive:DeleteInput
	medialive:DeleteInputSecurityGroup
	medialive:DeleteMultiplex
	medialive:DeleteMultiplexProgram
	medialive:DeleteNetwork
	medialive:DeleteNode
	medialive:DeleteReservation
	medialive:DeleteSchedule
	medialive:DeleteSdiSource
	medialive:DeleteSignalMap
	medialive:DescribeAccountConfiguration

Service prefix	Actions
	medialive:DescribeChannel
	medialive:DescribeChannelPlacementGroup
	medialive:DescribeCluster
	medialive:DescribeInput
	medialive:DescribeInputDevice
	medialive:DescribeInputDeviceThumbnail
	medialive:DescribeInputSecurityGroup
	medialive:DescribeMultiplex
	medialive:DescribeMultiplexProgram
	medialive:DescribeNetwork
	medialive:DescribeNode
	medialive:DescribeOffering
	medialive:DescribeReservation
	medialive:DescribeSchedule
	medialive:DescribeSdiSource
	medialive:DescribeThumbnails
	medialive:GetCloudWatchAlarmTemplate
	medialive:GetCloudWatchAlarmTemplateGroup
	medialive:GetEventBridgeRuleTemplate
	medialive:GetEventBridgeRuleTemplateGroup
	medialive:GetSignalMap

Service prefix	Actions
	medialive:ListAlerts
	medialive:ListChannelPlacementGroups
	medialive:ListChannels
	medialive:ListCloudWatchAlarmTemplateGroups
	medialive:ListCloudWatchAlarmTemplates
	medialive:ListClusterAlerts
	medialive:ListClusters
	medialive:ListEventBridgeRuleTemplateGroups
	medialive:ListEventBridgeRuleTemplates
	medialive:ListInputDeviceTransfers
	medialive:ListInputDevices
	medialive:ListInputSecurityGroups
	medialive:ListInputs
	medialive:ListMultiplexAlerts
	medialive:ListMultiplexPrograms
	medialive:ListMultiplexes
	medialive:ListNetworks
	medialive:ListNodes
	medialive:ListOfferings
	medialive:ListReservations
	medialive:ListSdiSources

Service prefix	Actions
	medialive:ListSignalMaps
	medialive:ListVersions
	medialive:PurchaseOffering
	medialive:RebootInputDevice
	medialive:RejectInputDeviceTransfer
	medialive:RestartChannelPipelines
	medialive:StartChannel
	medialive:StartDeleteMonitorDeployment
	medialive:StartInputDevice
	medialive:StartInputDeviceMaintenanceWindow
	medialive:StartMonitorDeployment
	medialive:StartMultiplex
	medialive:StartUpdateSignalMap
	medialive:StopChannel
	medialive:StopInputDevice
	medialive:StopMultiplex
	medialive:TransferInputDevice
	medialive:UpdateAccountConfiguration
	medialive:UpdateChannel
	medialive:UpdateChannelClass
	medialive:UpdateChannelPlacementGroup

Service prefix	Actions
	medialive:UpdateCloudWatchAlarmTemplate
	medialive:UpdateCloudWatchAlarmTemplateGroup
	medialive:UpdateCluster
	medialive:UpdateEventBridgeRuleTemplate
	medialive:UpdateEventBridgeRuleTemplateGroup
	medialive:UpdateInput
	medialive:UpdateInputDevice
	medialive:UpdateInputSecurityGroup
	medialive:UpdateMultiplex
	medialive:UpdateMultiplexProgram
	medialive:UpdateNetwork
	medialive:UpdateNode
	medialive:UpdateNodeState
	medialive:UpdateReservation
	medialive:UpdateSdiSource

Service prefix	Actions
mediatailor	mediatailor:ConfigureLogsForPlaybackConfiguration mediatailor:CreateChannel mediatailor:CreateLiveSource mediatailor:CreatePrefetchSchedule mediatailor:CreateProgram mediatailor:CreateSourceLocation mediatailor:CreateVodSource mediatailor:DeleteChannel mediatailor:DeleteChannelPolicy mediatailor:DeleteLiveSource mediatailor:DeletePlaybackConfiguration mediatailor:DeletePrefetchSchedule mediatailor:DeleteProgram mediatailor:DeleteSourceLocation mediatailor:DeleteVodSource mediatailor:DescribeChannel mediatailor:DescribeLiveSource mediatailor:DescribeProgram mediatailor:DescribeSourceLocation mediatailor:DescribeVodSource mediatailor:GetChannelPolicy

Service prefix	Actions
	mediatailor:GetChannelSchedule mediatailor:GetPlaybackConfiguration mediatailor:GetPrefetchSchedule mediatailor:ListAlerts mediatailor:ListChannels mediatailor:ListLiveSources mediatailor:ListPlaybackConfigurations mediatailor:ListPrefetchSchedules mediatailor:ListSourceLocations mediatailor:ListVodSources mediatailor:PutChannelPolicy mediatailor:PutPlaybackConfiguration mediatailor:StartChannel mediatailor:StopChannel mediatailor:UpdateChannel mediatailor:UpdateLiveSource mediatailor:UpdateProgram mediatailor:UpdateSourceLocation mediatailor:UpdateVodSource

Service prefix	Actions
memorydb	memorydb:BatchUpdateCluster
	memorydb:CopySnapshot
	memorydb:CreateAcl
	memorydb:CreateCluster
	memorydb:CreateMultiRegionCluster
	memorydb:CreateParameterGroup
	memorydb:CreateSnapshot
	memorydb:CreateSubnetGroup
	memorydb:CreateUser
	memorydb:DeleteAcl
	memorydb:DeleteCluster
	memorydb:DeleteMultiRegionCluster
	memorydb:DeleteParameterGroup
	memorydb:DeleteSnapshot
	memorydb:DeleteSubnetGroup
	memorydb:DeleteUser
	memorydb:DescribeAcls
	memorydb:DescribeClusters
	memorydb:DescribeEngineVersions
	memorydb:DescribeEvents
	memorydb:DescribeMultiRegionClusters

Service prefix	Actions
	memorydb:DescribeMultiRegionParameterGroups
	memorydb:DescribeMultiRegionParameters
	memorydb:DescribeParameterGroups
	memorydb:DescribeParameters
	memorydb:DescribeReservedNodes
	memorydb:DescribeReservedNodesOfferings
	memorydb:DescribeServiceUpdates
	memorydb:DescribeSnapshots
	memorydb:DescribeSubnetGroups
	memorydb:DescribeUsers
	memorydb:FailoverShard
	memorydb:ListAllowedMultiRegionClusterUpdates
	memorydb:ListAllowedNodeTypeUpdates
	memorydb:PurchaseReservedNodesOffering
	memorydb:ResetParameterGroup
	memorydb:UpdateAcl
	memorydb:UpdateCluster
	memorydb:UpdateMultiRegionCluster
	memorydb:UpdateParameterGroup
	memorydb:UpdateSubnetGroup
	memorydb:UpdateUser

Service prefix	Actions
mgh	mgh:AssociateCreatedArtifact mgh:AssociateDiscoveredResource mgh:AssociateSourceResource mgh:CreateHomeRegionControl mgh:CreateProgressUpdateStream mgh:DeleteHomeRegionControl mgh:DeleteProgressUpdateStream mgh:DescribeApplicationState mgh:DescribeHomeRegionControls mgh:DescribeMigrationTask mgh:DisassociateCreatedArtifact mgh:DisassociateDiscoveredResource mgh:DisassociateSourceResource mgh:GetHomeRegion mgh:ImportMigrationTask mgh:ListApplicationStates mgh:ListCreatedArtifacts mgh:ListDiscoveredResources mgh:ListMigrationTaskUpdates mgh:ListMigrationTasks mgh:ListProgressUpdateStreams

Service prefix	Actions
	mgh:ListSourceResources mgh:NotifyApplicationState mgh:NotifyMigrationTaskState mgh:PutResourceAttributes

Service prefix	Actions
mgn	mgn:ArchiveApplication
	mgn:ArchiveWave
	mgn:AssociateApplications
	mgn:AssociateSourceServers
	mgn:ChangeServerLifeCycleState
	mgn:CreateApplication
	mgn:CreateConnector
	mgn:CreateLaunchConfigurationTemplate
	mgn:CreateNetworkMigrationDefinition
	mgn:CreateReplicationConfigurationTemplate
	mgn:CreateWave
	mgn:DeleteApplication
	mgn:DeleteConnector
	mgn:DeleteJob
	mgn:DeleteLaunchConfigurationTemplate
	mgn:DeleteNetworkMigrationDefinition
	mgn:DeleteReplicationConfigurationTemplate
	mgn:DeleteSourceServer
	mgn:DeleteVcenterClient
	mgn:DeleteWave
	mgn:DescribeJobLogItems

Service prefix	Actions
	mgn:DescribeJobs
	mgn:DescribeLaunchConfigurationTemplates
	mgn:DescribeReplicationConfigurationTemplates
	mgn:DescribeVcenterClients
	mgn:DisassociateApplications
	mgn:DisassociateSourceServers
	mgn:DisconnectFromService
	mgn:FinalizeCutover
	mgn:GetNetworkMigrationDefinition
	mgn:GetNetworkMigrationMapperSegmentConstruct
	mgn:GetReplicationConfiguration
	mgn:InitializeService
	mgn:ListConnectors
	mgn:ListExportErrors
	mgn:ListExports
	mgn:ListImportErrors
	mgn:ListImportFileEnrichments
	mgn:ListImports
	mgn:ListManagedAccounts
	mgn:ListNetworkMigrationAnalyses
	mgn:ListNetworkMigrationAnalysisResults

Service prefix	Actions
	mgn:ListNetworkMigrationCodeGenerationSegments
	mgn:ListNetworkMigrationCodeGenerations
	mgn:ListNetworkMigrationDefinitions
	mgn:ListNetworkMigrationDeployedStacks
	mgn:ListNetworkMigrationDeployments
	mgn:ListNetworkMigrationExecutions
	mgn:ListNetworkMigrationMapperSegmentConstructs
	mgn:ListNetworkMigrationMapperSegments
	mgn:ListNetworkMigrationMappingUpdates
	mgn:ListNetworkMigrationMappings
	mgn:ListSourceServerActions
	mgn:ListTemplateActions
	mgn:MarkAsArchived
	mgn:PauseReplication
	mgn:PutSourceServerAction
	mgn:PutTemplateAction
	mgn:RemoveSourceServerAction
	mgn:RemoveTemplateAction
	mgn:ResumeReplication
	mgn:RetryDataReplication
	mgn:StartCutover

Service prefix	Actions
	mgn:StartExport
	mgn:StartImport
	mgn:StartImportFileEnrichment
	mgn:StartNetworkMigrationCodeGeneration
	mgn:StartNetworkMigrationMapping
	mgn:StartNetworkMigrationMappingUpdate
	mgn:StartReplication
	mgn:StartTest
	mgn:StopReplication
	mgn:TerminateTargetInstances
	mgn:UnarchiveApplication
	mgn:UnarchiveWave
	mgn:UpdateApplication
	mgn:UpdateConnector
	mgn:UpdateLaunchConfigurationTemplate
	mgn:UpdateNetworkMigrationDefinition
	mgn:UpdateNetworkMigrationMapperSegment
	mgn:UpdateReplicationConfiguration
	mgn:UpdateReplicationConfigurationTemplate
	mgn:UpdateSourceServer
	mgn:UpdateSourceServerReplicationType

Service prefix	Actions
	mgn:UpdateWave

Service prefix	Actions
migrationhub-strategy	migrationhub-strategy:GetAntiPattern migrationhub-strategy:GetApplicationComponentDetails migrationhub-strategy:GetApplicationComponentStrategies migrationhub-strategy:GetAssessment migrationhub-strategy:GetImportFileTask migrationhub-strategy:GetLatestAssessmentId migrationhub-strategy:GetMessage migrationhub-strategy:GetPortfolioPreferences migrationhub-strategy:GetPortfolioSummary migrationhub-strategy:GetRecommendationReportDetails migrationhub-strategy:GetServerDetails migrationhub-strategy:GetServerStrategies migrationhub-strategy:ListAnalyzableServers migrationhub-strategy:ListAntiPatterns migrationhub-strategy:ListApplicationComponents migrationhub-strategy:ListCollectors migrationhub-strategy:ListImportFileTask migrationhub-strategy:ListJarArtifacts migrationhub-strategy:ListServers migrationhub-strategy:PutLogData migrationhub-strategy:PutMetricData

Service prefix	Actions
	migrationhub-strategy:PutPortfolioPreferences migrationhub-strategy:RegisterCollector migrationhub-strategy:SendMessage migrationhub-strategy:StartAssessment migrationhub-strategy:StartImportFileTask migrationhub-strategy:StartRecommendationReportGeneration migrationhub-strategy:StopAssessment migrationhub-strategy:UpdateApplicationComponentConfig migrationhub-strategy:UpdateCollectorConfiguration migrationhub-strategy:UpdateServerConfig

Service prefix	Actions
mobiletargeting	mobiletargeting:CreateApp
	mobiletargeting:CreateCampaign
	mobiletargeting:CreateEmailTemplate
	mobiletargeting:CreateExportJob
	mobiletargeting:CreateImportJob
	mobiletargeting:CreateInAppTemplate
	mobiletargeting:CreateJourney
	mobiletargeting:CreatePushTemplate
	mobiletargeting:CreateRecommenderConfiguration
	mobiletargeting:CreateSegment
	mobiletargeting:CreateSmsTemplate
	mobiletargeting:CreateVoiceTemplate
	mobiletargeting:DeleteAdmChannel
	mobiletargeting:DeleteApnsChannel
	mobiletargeting:DeleteApnsSandboxChannel
	mobiletargeting:DeleteApnsVoipChannel
	mobiletargeting:DeleteApnsVoipSandboxChannel
	mobiletargeting:DeleteApp
	mobiletargeting:DeleteBaiduChannel
	mobiletargeting:DeleteCampaign
	mobiletargeting:DeleteEmailChannel

Service prefix	Actions
	mobiletargeting:DeleteEmailTemplate
	mobiletargeting:DeleteEndpoint
	mobiletargeting:DeleteEventStream
	mobiletargeting:DeleteGcmChannel
	mobiletargeting:DeleteInAppTemplate
	mobiletargeting:DeleteJourney
	mobiletargeting:DeletePushTemplate
	mobiletargeting:DeleteRecommenderConfiguration
	mobiletargeting:DeleteSegment
	mobiletargeting:DeleteSmsChannel
	mobiletargeting:DeleteSmsTemplate
	mobiletargeting:DeleteUserEndpoints
	mobiletargeting:DeleteVoiceChannel
	mobiletargeting:DeleteVoiceTemplate
	mobiletargeting:GetAdmChannel
	mobiletargeting:GetApnsChannel
	mobiletargeting:GetApnsSandboxChannel
	mobiletargeting:GetApnsVoipChannel
	mobiletargeting:GetApnsVoipSandboxChannel
	mobiletargeting:GetApp
	mobiletargeting:GetApplicationDateRangeKpi

Service prefix	Actions
	mobiletargeting:GetApplicationSettings
	mobiletargeting:GetApps
	mobiletargeting:GetBaiduChannel
	mobiletargeting:GetCampaign
	mobiletargeting:GetCampaignActivities
	mobiletargeting:GetCampaignDateRangeKpi
	mobiletargeting:GetCampaignVersion
	mobiletargeting:GetCampaignVersions
	mobiletargeting:GetCampaigns
	mobiletargeting:GetChannels
	mobiletargeting:GetEmailChannel
	mobiletargeting:GetEmailTemplate
	mobiletargeting:GetEndpoint
	mobiletargeting:GetEventStream
	mobiletargeting:GetExportJob
	mobiletargeting:GetExportJobs
	mobiletargeting:GetGcmChannel
	mobiletargeting:GetImportJob
	mobiletargeting:GetImportJobs
	mobiletargeting:GetInAppMessages
	mobiletargeting:GetInAppTemplate

Service prefix	Actions
	mobiletargeting:GetJourney
	mobiletargeting:GetJourneyDateRangeKpi
	mobiletargeting:GetJourneyExecutionActivityMetrics
	mobiletargeting:GetJourneyExecutionMetrics
	mobiletargeting:GetJourneyRunExecutionActivityMetrics
	mobiletargeting:GetJourneyRunExecutionMetrics
	mobiletargeting:GetJourneyRuns
	mobiletargeting:GetPushTemplate
	mobiletargeting:GetRecommenderConfiguration
	mobiletargeting:GetRecommenderConfigurations
	mobiletargeting:GetSegment
	mobiletargeting:GetSegmentExportJobs
	mobiletargeting:GetSegmentImportJobs
	mobiletargeting:GetSegmentVersion
	mobiletargeting:GetSegmentVersions
	mobiletargeting:GetSegments
	mobiletargeting:GetSmsChannel
	mobiletargeting:GetSmsTemplate
	mobiletargeting:GetUserEndpoints
	mobiletargeting:GetVoiceChannel
	mobiletargeting:GetVoiceTemplate

Service prefix	Actions
	mobiletargeting:ListJourneys
	mobiletargeting:ListTemplateVersions
	mobiletargeting:ListTemplates
	mobiletargeting:PhoneNumberValidate
	mobiletargeting:PutEventStream
	mobiletargeting:RemoveAttributes
	mobiletargeting:UpdateAdmChannel
	mobiletargeting:UpdateApnsChannel
	mobiletargeting:UpdateApnsSandboxChannel
	mobiletargeting:UpdateApnsVoipChannel
	mobiletargeting:UpdateApnsVoipSandboxChannel
	mobiletargeting:UpdateApplicationSettings
	mobiletargeting:UpdateBaiduChannel
	mobiletargeting:UpdateCampaign
	mobiletargeting:UpdateEmailChannel
	mobiletargeting:UpdateEmailTemplate
	mobiletargeting:UpdateEndpoint
	mobiletargeting:UpdateEndpointsBatch
	mobiletargeting:UpdateGcmChannel
	mobiletargeting:UpdateInAppTemplate
	mobiletargeting:UpdateJourney

Service prefix	Actions
	mobiletargeting:UpdateJourneyState
	mobiletargeting:UpdatePushTemplate
	mobiletargeting:UpdateRecommenderConfiguration
	mobiletargeting:UpdateSegment
	mobiletargeting:UpdateSmsChannel
	mobiletargeting:UpdateSmsTemplate
	mobiletargeting:UpdateTemplateActiveVersion
	mobiletargeting:UpdateVoiceChannel
	mobiletargeting:UpdateVoiceTemplate
	mobiletargeting:VerifyOTPMessage

Service prefix	Actions
mq	mq:CreateBroker mq:CreateConfiguration mq:CreateUser mq:DeleteBroker mq:DeleteConfiguration mq:DeleteUser mq:DescribeBroker mq:DescribeBrokerEngineTypes mq:DescribeBrokerInstanceOptions mq:DescribeConfiguration mq:DescribeConfigurationRevision mq:DescribeSharedResources mq:DescribeUser mq:ListBrokers mq:ListConfigurationRevisions mq:ListConfigurations mq:ListUsers mq:Promote mq:RebootBroker mq:UpdateBroker mq:UpdateConfiguration

Service prefix	Actions
	mq:UpdateUser

Service prefix	Actions
networkmanager	networkmanager:AcceptAttachment
	networkmanager:AssociateConnectPeer
	networkmanager:AssociateCustomerGateway
	networkmanager:AssociateLink
	networkmanager:AssociateTransitGatewayConnectPeer
	networkmanager:CreateConnectAttachment
	networkmanager:CreateConnectPeer
	networkmanager:CreateConnection
	networkmanager:CreateCoreNetwork
	networkmanager:CreateDevice
	networkmanager:CreateDirectConnectGatewayAttachment
	networkmanager:CreateGlobalNetwork
	networkmanager:CreateLink
	networkmanager:CreateSite
	networkmanager:CreateSiteToSiteVpnAttachment
	networkmanager:CreateTransitGatewayPeering
	networkmanager:CreateTransitGatewayRouteTableAttachment
	networkmanager:CreateVpcAttachment
	networkmanager:DeleteAttachment
	networkmanager:DeleteConnectPeer
	networkmanager:DeleteConnection

Service prefix	Actions
	networkmanager:DeleteCoreNetwork
	networkmanager:DeleteCoreNetworkPolicyVersion
	networkmanager:DeleteDevice
	networkmanager:DeleteGlobalNetwork
	networkmanager:DeleteLink
	networkmanager:DeletePeering
	networkmanager:DeleteResourcePolicy
	networkmanager:DeleteSite
	networkmanager:DeregisterTransitGateway
	networkmanager:DescribeGlobalNetworks
	networkmanager:DisassociateConnectPeer
	networkmanager:DisassociateCustomerGateway
	networkmanager:DisassociateLink
	networkmanager:DisassociateTransitGatewayConnectPeer
	networkmanager:ExecuteCoreNetworkChangeSet
	networkmanager:GetConnectAttachment
	networkmanager:GetConnectPeer
	networkmanager:GetConnectPeerAssociations
	networkmanager:GetConnections
	networkmanager:GetCoreNetwork
	networkmanager:GetCoreNetworkChangeEvents

Service prefix	Actions
	networkmanager:GetCoreNetworkChangeSet
	networkmanager:GetCoreNetworkPolicy
	networkmanager:GetCustomerGatewayAssociations
	networkmanager:GetDevices
	networkmanager:GetLinkAssociations
	networkmanager:GetLinks
	networkmanager:GetNetworkResourceCounts
	networkmanager:GetNetworkResourceRelationships
	networkmanager:GetNetworkResources
	networkmanager:GetNetworkRoutes
	networkmanager:GetNetworkTelemetry
	networkmanager:GetResourcePolicy
	networkmanager:GetRouteAnalysis
	networkmanager:GetSiteToSiteVpnAttachment
	networkmanager:GetSites
	networkmanager:GetTransitGatewayConnectPeerAssociations
	networkmanager:GetTransitGatewayPeering
	networkmanager:GetTransitGatewayRegistrations
	networkmanager:GetTransitGatewayRouteTableAttachment
	networkmanager:GetVpcAttachment
	networkmanager:ListAttachmentRoutingPolicyAssociations

Service prefix	Actions
	networkmanager:ListAttachments
	networkmanager:ListConnectPeers
	networkmanager:ListCoreNetworkPolicyVersions
	networkmanager:ListCoreNetworkPrefixListAssociations
	networkmanager:ListCoreNetworkRoutingInformation
	networkmanager:ListCoreNetworks
	networkmanager:ListOrganizationServiceAccessStatus
	networkmanager:ListPeerings
	networkmanager:PutAttachmentRoutingPolicyLabel
	networkmanager:PutCoreNetworkPolicy
	networkmanager:PutResourcePolicy
	networkmanager:RegisterTransitGateway
	networkmanager:RejectAttachment
	networkmanager:RemoveAttachmentRoutingPolicyLabel
	networkmanager:RestoreCoreNetworkPolicyVersion
	networkmanager:StartOrganizationServiceAccessUpdate
	networkmanager:StartRouteAnalysis
	networkmanager:UpdateConnection
	networkmanager:UpdateCoreNetwork
	networkmanager:UpdateDevice
	networkmanager:UpdateDirectConnectGatewayAttachment

Service prefix	Actions
	networkmanager:UpdateGlobalNetwork networkmanager:UpdateLink networkmanager:UpdateNetworkResourceMetadata networkmanager:UpdateSite networkmanager:UpdateVpcAttachment

Service prefix	Actions
nimble	nimble:AcceptEulas nimble:CreateLaunchProfile nimble:CreateStreamingImage nimble:CreateStreamingSession nimble:CreateStreamingSessionStream nimble:CreateStudio nimble:CreateStudioComponent nimble:DeleteLaunchProfile nimble:DeleteLaunchProfileMember nimble:DeleteStreamingImage nimble:DeleteStreamingSession nimble:DeleteStudio nimble:DeleteStudioComponent nimble:DeleteStudioMember nimble:GetEula nimble:GetLaunchProfileDetails nimble:GetStreamingImage nimble:GetStreamingSession nimble:GetStreamingSessionBackup nimble:GetStreamingSessionStream nimble:GetStudio

Service prefix	Actions
	nimble:GetStudioComponent nimble:GetStudioMember nimble:ListEulas nimble:ListLaunchProfileMembers nimble:ListLaunchProfiles nimble:ListStreamingImages nimble:ListStreamingSessionBackups nimble:ListStreamingSessions nimble:ListStudioComponents nimble:ListStudioMembers nimble:ListStudios nimble:PutLaunchProfileMembers nimble:PutStudioMembers nimble:StartStreamingSession nimble:StartStudioSSOConfigurationRepair nimble:StopStreamingSession nimble:UpdateLaunchProfile nimble:UpdateLaunchProfileMember nimble:UpdateStreamingImage nimble:UpdateStudio nimble:UpdateStudioComponent

Service prefix	Actions
omics	omics:AbortMultipartReadSetUpload
	omics:AcceptShare
	omics:BatchDeleteReadSet
	omics:CancelAnnotationImportJob
	omics:CancelRun
	omics:CancelRunBatch
	omics:CancelVariantImportJob
	omics:CompleteMultipartReadSetUpload
	omics:CreateAnnotationStore
	omics:CreateAnnotationStoreVersion
	omics:CreateConfiguration
	omics:CreateMultipartReadSetUpload
	omics:CreateReferenceStore
	omics:CreateRunGroup
	omics:CreateSequenceStore
	omics:CreateShare
	omics:CreateVariantStore
	omics:CreateWorkflow
	omics:CreateWorkflowVersion
	omics:DeleteAnnotationStore
	omics:DeleteAnnotationStoreVersions

Service prefix	Actions
	omics:DeleteBatch
	omics:DeleteConfiguration
	omics:DeleteReference
	omics:DeleteReferenceStore
	omics:DeleteRun
	omics:DeleteRunBatch
	omics:DeleteRunGroup
	omics:DeleteSequenceStore
	omics:DeleteShare
	omics:DeleteVariantStore
	omics:DeleteWorkflow
	omics:DeleteWorkflowVersion
	omics:GetAnnotationImportJob
	omics:GetAnnotationStore
	omics:GetAnnotationStoreVersion
	omics:GetBatch
	omics:GetConfiguration
	omics:GetReadSet
	omics:GetReadSetActivationJob
	omics:GetReadSetExportJob
	omics:GetReadSetImportJob

Service prefix	Actions
	omics:GetReadSetMetadata
	omics:GetReference
	omics:GetReferenceImportJob
	omics:GetReferenceMetadata
	omics:GetReferenceStore
	omics:GetRun
	omics:GetRunGroup
	omics:GetRunTask
	omics:GetSequenceStore
	omics:GetShare
	omics:GetVariantImportJob
	omics:GetVariantStore
	omics:GetWorkflow
	omics:GetWorkflowVersion
	omics:ListAnnotationImportJobs
	omics:ListAnnotationStoreVersions
	omics:ListAnnotationStores
	omics:ListBatch
	omics:ListConfigurations
	omics:ListMultipartReadSetUploads
	omics:ListReadSetActivationJobs

Service prefix	Actions
	omics:ListReadSetExportJobs
	omics:ListReadSetImportJobs
	omics:ListReadSetUploadParts
	omics:ListReadSets
	omics:ListReferenceImportJobs
	omics:ListReferenceStores
	omics:ListReferences
	omics:ListRunGroups
	omics:ListRunTasks
	omics:ListRuns
	omics:ListRunsInBatch
	omics:ListSequenceStores
	omics:ListShares
	omics:ListVariantImportJobs
	omics:ListVariantStores
	omics:ListWorkflowVersions
	omics:ListWorkflows
	omics:StartAnnotationImportJob
	omics:StartReadSetActivationJob
	omics:StartReadSetExportJob
	omics:StartReadSetImportJob

Service prefix	Actions
	omics:StartReferenceImportJob omics:StartRun omics:StartRunBatch omics:StartVariantImportJob omics:UpdateAnnotationStore omics:UpdateAnnotationStoreVersion omics:UpdateRunGroup omics:UpdateVariantStore omics:UpdateWorkflow omics:UpdateWorkflowVersion omics:UploadReadSetPart

Service prefix	Actions
opsworks	opsworks:AssignInstance opsworks:AssignVolume opsworks:AssociateElasticIp opsworks:AttachElasticLoadBalancer opsworks:CloneStack opsworks:CreateApp opsworks:CreateDeployment opsworks:CreateInstance opsworks:CreateLayer opsworks:CreateStack opsworks:CreateUserProfile opsworks:DeleteApp opsworks:DeleteInstance opsworks:DeleteLayer opsworks:DeleteStack opsworks:DeleteUserProfile opsworks:DeregisterEcsCluster opsworks:DeregisterElasticIp opsworks:DeregisterInstance opsworks:DeregisterRdsDbInstance opsworks:DeregisterVolume

Service prefix	Actions
	opsworks:DescribeAgentVersions opsworks:DescribeApps opsworks:DescribeCommands opsworks:DescribeDeployments opsworks:DescribeEcsClusters opsworks:DescribeElasticIps opsworks:DescribeElasticLoadBalancers opsworks:DescribeInstances opsworks:DescribeLayers opsworks:DescribeLoadBasedAutoScaling opsworks:DescribeMyUserProfile opsworks:DescribeOperatingSystems opsworks:DescribePermissions opsworks:DescribeRaidArrays opsworks:DescribeRdsDbInstances opsworks:DescribeServiceErrors opsworks:DescribeStackProvisioningParameters opsworks:DescribeStackSummary opsworks:DescribeStacks opsworks:DescribeTimeBasedAutoScaling opsworks:DescribeUserProfiles

Service prefix	Actions
	opsworks:DescribeVolumes opsworks:DetachElasticLoadBalancer opsworks:DisassociateElasticIp opsworks:GetHostnameSuggestion opsworks:GrantAccess opsworks:RebootInstance opsworks:RegisterEcsCluster opsworks:RegisterElasticIp opsworks:RegisterInstance opsworks:RegisterRdsDbInstance opsworks:RegisterVolume opsworks:SetLoadBasedAutoScaling opsworks:SetPermission opsworks:SetTimeBasedAutoScaling opsworks:StartInstance opsworks:StartStack opsworks:StopInstance opsworks:StopStack opsworks:UnassignInstance opsworks:UnassignVolume opsworks:UpdateApp

Service prefix	Actions
	opsworks:UpdateElasticIp opsworks:UpdateInstance opsworks:UpdateLayer opsworks:UpdateMyUserProfile opsworks:UpdateRdsDbInstance opsworks:UpdateStack opsworks:UpdateUserProfile opsworks:UpdateVolume

Service prefix	Actions
opsworks-cm	opsworks-cm:AssociateNode
	opsworks-cm:CreateBackup
	opsworks-cm:CreateServer
	opsworks-cm:DeleteBackup
	opsworks-cm:DeleteServer
	opsworks-cm:DescribeAccountAttributes
	opsworks-cm:DescribeBackups
	opsworks-cm:DescribeEvents
	opsworks-cm:DescribeNodeAssociationStatus
	opsworks-cm:DescribeServers
	opsworks-cm:DisassociateNode
	opsworks-cm:ExportServerEngineAttribute
	opsworks-cm:RestoreServer
	opsworks-cm:StartMaintenance
	opsworks-cm:UpdateServer
	opsworks-cm:UpdateServerEngineAttributes

Service prefix	Actions
organizations	organizations:AcceptHandshake organizations:AttachPolicy organizations:CancelHandshake organizations:CloseAccount organizations:CreateAccount organizations:CreateGovCloudAccount organizations:CreateOrganization organizations:CreateOrganizationalUnit organizations:CreatePolicy organizations:DeclineHandshake organizations:DeleteOrganization organizations:DeleteOrganizationalUnit organizations:DeletePolicy organizations:DeleteResourcePolicy organizations:DeregisterDelegatedAdministrator organizations:DescribeAccount organizations:DescribeCreateAccountStatus organizations:DescribeEffectivePolicy organizations:DescribeHandshake organizations:DescribeOrganization organizations:DescribeOrganizationalUnit

Service prefix	Actions
	organizations:DescribePolicy organizations:DescribeResourcePolicy organizations:DescribeResponsibilityTransfer organizations:DetachPolicy organizations:DisableAWSServiceAccess organizations:DisablePolicyType organizations:EnableAWSServiceAccess organizations:EnableAllFeatures organizations:EnablePolicyType organizations:InviteAccountToOrganization organizations:LeaveOrganization organizations:ListAWSServiceAccessForOrganization organizations:ListAccounts organizations:ListAccountsForParent organizations:ListAccountsWithInvalidEffectivePolicy organizations:ListChildren organizations:ListCreateAccountStatus organizations:ListDelegatedAdministrators organizations:ListDelegatedServicesForAccount organizations:ListHandshakesForAccount organizations:ListHandshakesForOrganization

Service prefix	Actions
	organizations:ListInboundResponsibilityTransfers organizations:ListOrganizationalUnitsForParent organizations:ListOutboundResponsibilityTransfers organizations:ListParents organizations:ListPolicies organizations:ListPoliciesForTarget organizations:ListRoots organizations:ListTargetsForPolicy organizations:MoveAccount organizations:PutResourcePolicy organizations:RegisterDelegatedAdministrator organizations:RemoveAccountFromOrganization organizations:TerminateResponsibilityTransfer organizations:UpdateOrganizationalUnit organizations:UpdatePolicy organizations:UpdateResponsibilityTransfer

Service prefix	Actions
outposts	outposts:CancelCapacityTask
	outposts:CancelOrder
	outposts:CreateOrder
	outposts:CreateOutpost
	outposts:CreatePrivateConnectivityConfig
	outposts:CreateQuote
	outposts:CreateRenewal
	outposts:CreateSite
	outposts:DeleteOutpost
	outposts:DeleteQuote
	outposts:DeleteSite
	outposts:GetCapacityTask
	outposts:GetCatalogItem
	outposts:GetConnection
	outposts:GetOrder
	outposts:GetOutpost
	outposts:GetOutpostBillingInformation
	outposts:GetOutpostInstanceTypes
	outposts:GetOutpostSupportedInstanceTypes
	outposts:GetPrivateConnectivityConfig
	outposts:GetQuote

Service prefix	Actions
	outposts:GetRenewalPricing
	outposts:GetSite
	outposts:GetSiteAddress
	outposts:ListAssetInstances
	outposts:ListAssets
	outposts:ListBlockingInstancesForCapacityTask
	outposts:ListCapacityTasks
	outposts:ListCatalogItems
	outposts:ListOrderableInstanceTypes
	outposts:ListOrders
	outposts:ListOutposts
	outposts:ListQuotes
	outposts:ListSites
	outposts:StartCapacityTask
	outposts:StartConnection
	outposts:UpdateOutpost
	outposts:UpdateQuote
	outposts:UpdateSite
	outposts:UpdateSiteAddress
	outposts:UpdateSiteRackPhysicalProperties

Service prefix	Actions
pi	pi:CreatePerformanceAnalysisReport pi:DeletePerformanceAnalysisReport pi:DescribeDimensionKeys pi:GetDimensionKeyDetails pi:GetPerformanceAnalysisReport pi:GetResourceMetadata pi:GetResourceMetrics pi:ListAvailableResourceDimensions pi:ListAvailableResourceMetrics pi:ListPerformanceAnalysisReports
pipes	pipes:CreatePipe pipes:DeletePipe pipes:DescribePipe pipes:ListPipes pipes:StartPipe pipes:StopPipe pipes:UpdatePipe

Service prefix	Actions
polly	polly:DeleteLexicon polly:DescribeVoices polly:GetLexicon polly:GetSpeechSynthesisTask polly:ListLexicons polly:ListSpeechSynthesisTasks polly:PutLexicon polly:StartSpeechSynthesisTask polly:SynthesizeSpeech

Service prefix	Actions
profile	profile:AddProfileKey
	profile:BatchGetCalculatedAttributeForProfile
	profile:BatchGetProfile
	profile:CreateCalculatedAttributeDefinition
	profile:CreateDomain
	profile:CreateEventStream
	profile:CreateProfile
	profile:CreateRecommender
	profile:CreateRecommenderFilter
	profile:CreateRecommenderSchema
	profile:CreateSegmentDefinition
	profile:CreateSegmentEstimate
	profile:CreateSegmentSnapshot
	profile:CreateUploadJob
	profile:DeleteCalculatedAttributeDefinition
	profile:DeleteDomain
	profile:DeleteDomainObjectType
	profile:DeleteEventStream
	profile:DeleteIntegration
	profile:DeleteProfile
	profile:DeleteProfileKey

Service prefix	Actions
	profile:DeleteProfileObject
	profile:DeleteProfileObjectType
	profile:DeleteRecommender
	profile:DeleteRecommenderFilter
	profile:DeleteRecommenderSchema
	profile:DeleteSegmentDefinition
	profile:DeleteWorkflow
	profile:DetectProfileObjectType
	profile:GetAutoMergingPreview
	profile:GetCalculatedAttributeDefinition
	profile:GetCalculatedAttributeForProfile
	profile:GetDomain
	profile:GetDomainObjectType
	profile:GetEventStream
	profile:GetIdentityResolutionJob
	profile:GetIntegration
	profile:GetMatches
	profile:GetObjectTypeAttributeStatistics
	profile:GetProfileObjectType
	profile:GetProfileObjectTypeTemplate
	profile:GetProfileRecommendations

Service prefix	Actions
	profile:GetRecommender
	profile:GetRecommenderFilter
	profile:GetRecommenderSchema
	profile:GetSegmentDefinition
	profile:GetSegmentEstimate
	profile:GetSegmentMembership
	profile:GetSegmentSnapshot
	profile:GetSimilarProfiles
	profile:GetUploadJob
	profile:GetUploadJobPath
	profile:GetWorkflow
	profile:GetWorkflowSteps
	profile:ListAccountIntegrations
	profile:ListCalculatedAttributeDefinitions
	profile:ListCalculatedAttributesForProfile
	profile:ListDomainLayouts
	profile:ListDomainObjectTypes
	profile:ListDomains
	profile:ListEventStreams
	profile:ListIdentityResolutionJobs
	profile:ListIntegrations

Service prefix	Actions
	profile:ListObjectTypeAttributeValues
	profile:ListObjectTypeAttributes
	profile:ListProfileAttributeValues
	profile:ListProfileObjectTypeTemplates
	profile:ListProfileObjectTypes
	profile:ListProfileObjects
	profile:ListRecommenderFilters
	profile:ListRecommenderRecipes
	profile:ListRecommenderSchemas
	profile:ListRecommenders
	profile:ListRuleBasedMatches
	profile:ListSegmentDefinitions
	profile:ListUploadJobs
	profile:ListWorkflows
	profile:MergeProfiles
	profile:PutDomainObjectType
	profile:PutIntegration
	profile:PutProfileObject
	profile:PutProfileObjectType
	profile:SearchProfiles
	profile:StartRecommender

Service prefix	Actions
	profile:StartUploadJob
	profile:StopRecommender
	profile:StopUploadJob
	profile:UpdateCalculatedAttributeDefinition
	profile:UpdateDomain
	profile:UpdateProfile
	profile:UpdateRecommender

Service prefix	Actions
qldb	qldb:CancelJournalKinesisStream qldb:CreateLedger qldb:DeleteLedger qldb:DescribeJournalKinesisStream qldb:DescribeJournalS3Export qldb:DescribeLedger qldb:ExportJournalToS3 qldb:GetBlock qldb:GetDigest qldb:GetRevision qldb:ListJournalKinesisStreamsForLedger qldb:ListJournalS3Exports qldb:ListJournalS3ExportsForLedger qldb:ListLedgers qldb:StreamJournalToKinesis qldb:UpdateLedger qldb:UpdateLedgerPermissionsMode

Service prefix	Actions
ram	ram:AcceptResourceShareInvitation
	ram:AssociateResourceShare
	ram:AssociateResourceSharePermission
	ram:CreatePermission
	ram:CreatePermissionVersion
	ram:CreateResourceShare
	ram:DeletePermission
	ram:DeletePermissionVersion
	ram:DeleteResourceShare
	ram:DisassociateResourceShare
	ram:DisassociateResourceSharePermission
	ram:EnableSharingWithAwsOrganization
	ram:GetPermission
	ram:GetResourcePolicies
	ram:GetResourceShareAssociations
	ram:GetResourceShareInvitations
	ram:GetResourceShares
	ram:ListPendingInvitationResources
	ram:ListPermissionAssociations
	ram:ListPermissionVersions
	ram:ListPermissions

Service prefix	Actions
	ram:ListPrincipals ram:ListReplacePermissionAssociationsWork ram:ListResourceSharePermissions ram:ListResourceTypes ram:ListResources ram:ListSourceAssociations ram:PromotePermissionCreatedFromPolicy ram:PromoteResourceShareCreatedFromPolicy ram:RejectResourceShareInvitation ram:ReplacePermissionAssociations ram:SetDefaultPermissionVersion ram:UpdateResourceShare
rbin	rbin:CreateRule rbin:DeleteRule rbin:GetRule rbin:ListRules rbin:LockRule rbin:UnlockRule rbin:UpdateRule

Service prefix	Actions
rds	rds:AddRoleToDBCluster
	rds:AddRoleToDBInstance
	rds:AddSourceIdentifierToSubscription
	rds:ApplyPendingMaintenanceAction
	rds:AuthorizeDBSecurityGroupIngress
	rds:BacktrackDBCluster
	rds:CancelExportTask
	rds:CopyDBClusterParameterGroup
	rds:CopyDBClusterSnapshot
	rds:CopyDBParameterGroup
	rds:CopyDBSnapshot
	rds:CopyOptionGroup
	rds:CreateCustomDBEngineVersion
	rds:CreateDBClusterParameterGroup
	rds:CreateDBParameterGroup
	rds:CreateDBProxy
	rds:CreateDBProxyEndpoint
	rds:CreateDBSecurityGroup
	rds:CreateDBSubnetGroup
	rds:CreateEventSubscription
	rds:CreateGlobalCluster

Service prefix	Actions
	rds:CreateOptionGroup
	rds:DeleteBlueGreenDeployment
	rds:DeleteDBClusterAutomatedBackup
	rds:DeleteDBClusterParameterGroup
	rds:DeleteDBClusterSnapshot
	rds:DeleteDBInstanceAutomatedBackup
	rds:DeleteDBParameterGroup
	rds:DeleteDBProxy
	rds:DeleteDBProxyEndpoint
	rds:DeleteDBSecurityGroup
	rds:DeleteDBSnapshot
	rds:DeleteDBSubnetGroup
	rds:DeleteEventSubscription
	rds:DeleteGlobalCluster
	rds:DeleteOptionGroup
	rds:DeregisterDBProxyTargets
	rds:DescribeAccountAttributes
	rds:DescribeBlueGreenDeployments
	rds:DescribeCertificates
	rds:DescribeDBClusterAutomatedBackups
	rds:DescribeDBClusterBacktracks

Service prefix	Actions
	rds:DescribeDBClusterEndpoints
	rds:DescribeDBClusterParameterGroups
	rds:DescribeDBClusterParameters
	rds:DescribeDBClusterSnapshotAttributes
	rds:DescribeDBClusterSnapshots
	rds:DescribeDBClusters
	rds:DescribeDBEngineVersions
	rds:DescribeDBInstanceAutomatedBackups
	rds:DescribeDBInstances
	rds:DescribeDBLogFiles
	rds:DescribeDBMajorEngineVersions
	rds:DescribeDBParameterGroups
	rds:DescribeDBParameters
	rds:DescribeDBProxies
	rds:DescribeDBProxyEndpoints
	rds:DescribeDBProxyTargetGroups
	rds:DescribeDBProxyTargets
	rds:DescribeDBRecommendations
	rds:DescribeDBSecurityGroups
	rds:DescribeDBSnapshotAttributes
	rds:DescribeDBSnapshotTenantDatabases

Service prefix	Actions
	rds:DescribeDBSnapshots
	rds:DescribeDBSubnetGroups
	rds:DescribeEngineDefaultClusterParameters
	rds:DescribeEngineDefaultParameters
	rds:DescribeEventCategories
	rds:DescribeEventSubscriptions
	rds:DescribeEvents
	rds:DescribeExportTasks
	rds:DescribeGlobalClusters
	rds:DescribeIntegrations
	rds:DescribeOptionGroupOptions
	rds:DescribeOptionGroups
	rds:DescribeOrderableDBInstanceOptions
	rds:DescribePendingMaintenanceActions
	rds:DescribeReservedDBInstances
	rds:DescribeReservedDBInstancesOfferings
	rds:DescribeSourceRegions
	rds:DescribeTenantDatabases
	rds:DescribeValidDBInstanceModifications
	rds:DownloadCompleteDBLogFile
	rds:DownloadDBLogFilePortion

Service prefix	Actions
	rds:FailoverDBCluster
	rds:FailoverGlobalCluster
	rds:ModifyActivityStream
	rds:ModifyCertificates
	rds:ModifyCurrentDBClusterCapacity
	rds:ModifyDBClusterEndpoint
	rds:ModifyDBClusterParameterGroup
	rds:ModifyDBClusterSnapshotAttribute
	rds:ModifyDBParameterGroup
	rds:ModifyDBProxy
	rds:ModifyDBProxyEndpoint
	rds:ModifyDBProxyTargetGroup
	rds:ModifyDBRecommendation
	rds:ModifyDBSnapshot
	rds:ModifyDBSnapshotAttribute
	rds:ModifyDBSubnetGroup
	rds:ModifyEventSubscription
	rds:ModifyGlobalCluster
	rds:ModifyOptionGroup
	rds:ModifyTenantDatabase
	rds:PurchaseReservedDBInstancesOffering

Service prefix	Actions
	rds:RebootDBCluster
	rds:RegisterDBProxyTargets
	rds:RemoveFromGlobalCluster
	rds:RemoveRoleFromDBCluster
	rds:RemoveRoleFromDBInstance
	rds:RemoveSourceIdentifierFromSubscription
	rds:ResetDBClusterParameterGroup
	rds:ResetDBParameterGroup
	rds:RestoreDBClusterFromS3
	rds:RestoreDBClusterFromSnapshot
	rds:RestoreDBClusterToPointInTime
	rds:RestoreDBInstanceFromDBSnapshot
	rds:RestoreDBInstanceFromS3
	rds:RestoreDBInstanceToPointInTime
	rds:RevokeDBSecurityGroupIngress
	rds:StartActivityStream
	rds:StartDBCluster
	rds:StartDBInstance
	rds:StartDBInstanceAutomatedBackupsReplication
	rds:StartExportTask
	rds:StopActivityStream

Service prefix	Actions
	rds:StopDBCluster rds:StopDBInstance rds:StopDBInstanceAutomatedBackupsReplication rds:SwitchoverBlueGreenDeployment rds:SwitchoverGlobalCluster rds:SwitchoverReadReplica

Service prefix	Actions
redshift	redshift:AcceptReservedNodeExchange redshift:AddPartner redshift:AssociateDataShareConsumer redshift:AuthorizeClusterSecurityGroupIngress redshift:AuthorizeDataShare redshift:AuthorizeEndpointAccess redshift:AuthorizeSnapshotAccess redshift:BatchDeleteClusterSnapshots redshift:BatchModifyClusterSnapshots redshift:CancelResize redshift:CopyClusterSnapshot redshift:CreateAuthenticationProfile redshift:CreateCluster redshift:CreateClusterParameterGroup redshift:CreateClusterSecurityGroup redshift:CreateClusterSnapshot redshift:CreateClusterSubnetGroup redshift:CreateCustomDomainAssociation redshift:CreateEndpointAccess redshift:CreateEventSubscription redshift:CreateHsmClientCertificate

Service prefix	Actions
	redshift:CreateHsmConfiguration redshift:CreateIntegration redshift:CreateRedshiftIdcApplication redshift:CreateScheduledAction redshift:CreateSnapshotCopyGrant redshift:CreateSnapshotSchedule redshift:CreateUsageLimit redshift:DeauthorizeDataShare redshift:DeleteAuthenticationProfile redshift:DeleteCluster redshift:DeleteClusterParameterGroup redshift:DeleteClusterSecurityGroup redshift:DeleteClusterSnapshot redshift:DeleteClusterSubnetGroup redshift:DeleteCustomDomainAssociation redshift:DeleteEndpointAccess redshift:DeleteEventSubscription redshift:DeleteHsmClientCertificate redshift:DeleteHsmConfiguration redshift:DeletePartner redshift:DeleteRedshiftIdcApplication

Service prefix	Actions
	redshift:DeleteResourcePolicy redshift:DeleteScheduledAction redshift:DeleteSnapshotCopyGrant redshift:DeleteSnapshotSchedule redshift:DeleteUsageLimit redshift:DeregisterNamespace redshift:DescribeAccountAttributes redshift:DescribeAuthenticationProfiles redshift:DescribeClusterDbRevisions redshift:DescribeClusterParameterGroups redshift:DescribeClusterParameters redshift:DescribeClusterSecurityGroups redshift:DescribeClusterSnapshots redshift:DescribeClusterSubnetGroups redshift:DescribeClusterTracks redshift:DescribeClusterVersions redshift:DescribeClusters redshift:DescribeCustomDomainAssociations redshift:DescribeDataShares redshift:DescribeDataSharesForConsumer redshift:DescribeDataSharesForProducer

Service prefix	Actions
	redshift:DescribeDefaultClusterParameters redshift:DescribeEndpointAccess redshift:DescribeEndpointAuthorization redshift:DescribeEventCategories redshift:DescribeEventSubscriptions redshift:DescribeEvents redshift:DescribeHsmClientCertificates redshift:DescribeHsmConfigurations redshift:DescribeInboundIntegrations redshift:DescribeIntegrations redshift:DescribeLoggingStatus redshift:DescribeNodeConfigurationOptions redshift:DescribeOrderableClusterOptions redshift:DescribePartners redshift:DescribeRedshiftIdcApplications redshift:DescribeReservedNodeExchangeStatus redshift:DescribeReservedNodeOfferings redshift:DescribeReservedNodes redshift:DescribeResize redshift:DescribeScheduledActions redshift:DescribeSnapshotCopyGrants

Service prefix	Actions
	redshift:DescribeSnapshotSchedules redshift:DescribeStorage redshift:DescribeTableRestoreStatus redshift:DescribeUsageLimits redshift:DisableLogging redshift:DisableSnapshotCopy redshift:DisassociateDataShareConsumer redshift:EnableLogging redshift:EnableSnapshotCopy redshift:FailoverPrimaryCompute redshift:GetClusterCredentials redshift:GetClusterCredentialsWithIAM redshift:GetIdentityCenterAuthToken redshift:GetReservedNodeExchangeConfigurationOptions redshift:GetReservedNodeExchangeOfferings redshift:GetResourcePolicy redshift:ListRecommendations redshift:ModifyAquaConfiguration redshift:ModifyAuthenticationProfile redshift:ModifyCluster redshift:ModifyClusterDbRevision

Service prefix	Actions
	redshift:ModifyClusterIamRoles redshift:ModifyClusterMaintenance redshift:ModifyClusterParameterGroup redshift:ModifyClusterSnapshot redshift:ModifyClusterSnapshotSchedule redshift:ModifyClusterSubnetGroup redshift:ModifyCustomDomainAssociation redshift:ModifyEndpointAccess redshift:ModifyEventSubscription redshift:ModifyRedshiftIdcApplication redshift:ModifyScheduledAction redshift:ModifySnapshotCopyRetentionPeriod redshift:ModifySnapshotSchedule redshift:ModifyUsageLimit redshift:PauseCluster redshift:PurchaseReservedNodeOffering redshift:PutResourcePolicy redshift:RebootCluster redshift:RegisterNamespace redshift:RejectDataShare redshift:ResetClusterParameterGroup

Service prefix	Actions
	redshift:ResizeCluster redshift:RestoreFromClusterSnapshot redshift:RestoreTableFromClusterSnapshot redshift:ResumeCluster redshift:RevokeClusterSecurityGroupIngress redshift:RevokeEndpointAccess redshift:RevokeSnapshotAccess redshift:RotateEncryptionKey redshift:UpdatePartnerStatus
redshift-data	redshift-data:BatchExecuteStatement redshift-data:CancelStatement redshift-data:DescribeStatement redshift-data:DescribeTable redshift-data:ExecuteStatement redshift-data:GetStatementResult redshift-data:ListDatabases redshift-data:ListSchemas redshift-data:ListSessions redshift-data:ListStatements redshift-data:ListTables

Service prefix	Actions
refactor-spaces	refactor-spaces:CreateApplication refactor-spaces:CreateEnvironment refactor-spaces:CreateRoute refactor-spaces:CreateService refactor-spaces:DeleteApplication refactor-spaces:DeleteEnvironment refactor-spaces:DeleteResourcePolicy refactor-spaces:DeleteRoute refactor-spaces:DeleteService refactor-spaces:GetApplication refactor-spaces:GetEnvironment refactor-spaces:GetResourcePolicy refactor-spaces:GetRoute refactor-spaces:GetService refactor-spaces:ListApplications refactor-spaces:ListEnvironmentVpcs refactor-spaces:ListEnvironments refactor-spaces:ListRoutes refactor-spaces:ListServices refactor-spaces:PutResourcePolicy refactor-spaces:UpdateRoute

Service prefix	Actions
rekognition	rekognition:AssociateFaces
	rekognition:CompareFaces
	rekognition:CopyProjectVersion
	rekognition:CreateCollection
	rekognition:CreateDataset
	rekognition:CreateFaceLivenessSession
	rekognition:CreateProject
	rekognition:CreateProjectVersion
	rekognition:CreateStreamProcessor
	rekognition:CreateUser
	rekognition:DeleteCollection
	rekognition:DeleteDataset
	rekognition:DeleteFaces
	rekognition:DeleteProject
	rekognition:DeleteProjectPolicy
	rekognition:DeleteProjectVersion
	rekognition:DeleteStreamProcessor
	rekognition:DeleteUser
	rekognition:DescribeCollection
	rekognition:DescribeDataset
	rekognition:DescribeProjectVersions

Service prefix	Actions
	rekognition:DescribeProjects
	rekognition:DescribeStreamProcessor
	rekognition:DetectCustomLabels
	rekognition:DetectFaces
	rekognition:DetectLabels
	rekognition:DetectModerationLabels
	rekognition:DetectProtectiveEquipment
	rekognition:DetectText
	rekognition:DisassociateFaces
	rekognition:DistributeDatasetEntries
	rekognition:GetCelebrityInfo
	rekognition:GetCelebrityRecognition
	rekognition:GetContentModeration
	rekognition:GetFaceDetection
	rekognition:GetFaceLivenessSessionResults
	rekognition:GetFaceSearch
	rekognition:GetLabelDetection
	rekognition:GetMediaAnalysisJob
	rekognition:GetPersonTracking
	rekognition:GetSegmentDetection
	rekognition:GetTextDetection

Service prefix	Actions
	rekognition:IndexFaces
	rekognition:ListCollections
	rekognition:ListDatasetEntries
	rekognition:ListDatasetLabels
	rekognition:ListFaces
	rekognition:ListMediaAnalysisJobs
	rekognition:ListProjectPolicies
	rekognition:ListStreamProcessors
	rekognition:ListUsers
	rekognition:PutProjectPolicy
	rekognition:RecognizeCelebrities
	rekognition:SearchFaces
	rekognition:SearchFacesByImage
	rekognition:SearchUsers
	rekognition:SearchUsersByImage
	rekognition:StartCelebrityRecognition
	rekognition:StartContentModeration
	rekognition:StartFaceDetection
	rekognition:StartFaceLivenessSession
	rekognition:StartFaceSearch
	rekognition:StartLabelDetection

Service prefix	Actions
	rekognition:StartMediaAnalysisJob rekognition:StartPersonTracking rekognition:StartProjectVersion rekognition:StartSegmentDetection rekognition:StartStreamProcessor rekognition:StartTextDetection rekognition:StopProjectVersion rekognition:StopStreamProcessor rekognition:UpdateDatasetEntries rekognition:UpdateStreamProcessor

Service prefix	Actions
resiliencehub	resiliencehub:AcceptResourceGroupingRecommendations resiliencehub:AddDraftAppVersionResourceMappings resiliencehub:BatchUpdateRecommendationStatus resiliencehub:CreateApp resiliencehub:CreateAppVersionAppComponent resiliencehub:CreateAppVersionResource resiliencehub:CreateRecommendationTemplate resiliencehub:CreateResiliencyPolicy resiliencehub:DeleteApp resiliencehub:DeleteAppAssessment resiliencehub:DeleteAppInputSource resiliencehub:DeleteAppVersionAppComponent resiliencehub:DeleteAppVersionResource resiliencehub:DeleteRecommendationTemplate resiliencehub:DeleteResiliencyPolicy resiliencehub:DescribeApp resiliencehub:DescribeAppAssessment resiliencehub:DescribeAppVersion resiliencehub:DescribeAppVersionAppComponent resiliencehub:DescribeAppVersionResource resiliencehub:DescribeAppVersionResourcesResolutionStatus

Service prefix	Actions
	resiliencehub:DescribeAppVersionTemplate resiliencehub:DescribeDraftAppVersionResourcesImportStatus resiliencehub:DescribeMetricsExport resiliencehub:DescribeResiliencyPolicy resiliencehub:DescribeResourceGroupingRecommendationTask resiliencehub:ImportResourcesToDraftAppVersion resiliencehub:ListAlarmRecommendations resiliencehub:ListAppAssessmentComplianceDrifts resiliencehub:ListAppAssessmentResourceDrifts resiliencehub:ListAppAssessments resiliencehub:ListAppComponentCompliances resiliencehub:ListAppComponentRecommendations resiliencehub:ListAppInputSources resiliencehub:ListAppVersionAppComponents resiliencehub:ListAppVersionResourceMappings resiliencehub:ListAppVersionResources resiliencehub:ListAppVersions resiliencehub:ListApps resiliencehub:ListMetrics resiliencehub:ListRecommendationTemplates resiliencehub:ListResiliencyPolicies

Service prefix	Actions
	resiliencehub:ListResourceGroupingRecommendations resiliencehub:ListSopRecommendations resiliencehub:ListSuggestedResiliencyPolicies resiliencehub:ListTestRecommendations resiliencehub:ListUnsupportedAppVersionResources resiliencehub:PublishAppVersion resiliencehub:PutDraftAppVersionTemplate resiliencehub:RejectResourceGroupingRecommendations resiliencehub:RemoveDraftAppVersionResourceMappings resiliencehub:ResolveAppVersionResources resiliencehub:StartAppAssessment resiliencehub:StartResourceGroupingRecommendationTask resiliencehub:UpdateApp resiliencehub:UpdateAppVersion resiliencehub:UpdateAppVersionAppComponent resiliencehub:UpdateAppVersionResource resiliencehub:UpdateResiliencyPolicy

Service prefix	Actions
resource-explorer-2	resource-explorer-2:AssociateDefaultView
	resource-explorer-2:BatchGetView
	resource-explorer-2:CreateIndex
	resource-explorer-2:CreateResourceExplorerSetup
	resource-explorer-2:CreateView
	resource-explorer-2:DeleteIndex
	resource-explorer-2:DeleteResourceExplorerSetup
	resource-explorer-2:DeleteView
	resource-explorer-2:DisassociateDefaultView
	resource-explorer-2:GetAccountLevelServiceConfiguration
	resource-explorer-2:GetDefaultView
	resource-explorer-2:GetIndex
	resource-explorer-2:GetManagedView
	resource-explorer-2:GetResourceExplorerSetup
	resource-explorer-2:GetServiceIndex
	resource-explorer-2:GetServiceView
	resource-explorer-2:ListIndexes
	resource-explorer-2:ListIndexesForMembers
	resource-explorer-2:ListManagedViews
	resource-explorer-2:ListServiceIndexes
	resource-explorer-2:ListServiceViews

Service prefix	Actions
	resource-explorer-2:ListStreamingAccessForServices resource-explorer-2:ListSupportedResourceTypes resource-explorer-2:ListViews resource-explorer-2:Search resource-explorer-2:UpdateIndexType resource-explorer-2:UpdateView

Service prefix	Actions
resource-groups	resource-groups:CancelTagSyncTask resource-groups:GetAccountSettings resource-groups:GetGroup resource-groups:GetGroupConfiguration resource-groups:GetGroupQuery resource-groups:GetTagSyncTask resource-groups:GroupResources resource-groups:ListGroupResources resource-groups:ListGroupingStatuses resource-groups:ListGroups resource-groups:ListTagSyncTasks resource-groups:PutGroupConfiguration resource-groups:SearchResources resource-groups:StartTagSyncTask resource-groups:UngroupResources resource-groups:UpdateAccountSettings resource-groups:UpdateGroup resource-groups:UpdateGroupQuery

Service prefix	Actions
robomaker	robomaker:BatchDeleteWorlds
	robomaker:BatchDescribeSimulationJob
	robomaker:CancelDeploymentJob
	robomaker:CancelSimulationJob
	robomaker:CancelSimulationJobBatch
	robomaker:CancelWorldExportJob
	robomaker:CancelWorldGenerationJob
	robomaker:CreateDeploymentJob
	robomaker:CreateFleet
	robomaker:CreateRobot
	robomaker:CreateRobotApplication
	robomaker:CreateRobotApplicationVersion
	robomaker:CreateSimulationApplication
	robomaker:CreateSimulationApplicationVersion
	robomaker:CreateSimulationJob
	robomaker:CreateWorldExportJob
	robomaker:CreateWorldGenerationJob
	robomaker:CreateWorldTemplate
	robomaker:DeleteFleet
	robomaker:DeleteRobot
	robomaker:DeleteRobotApplication

Service prefix	Actions
	robomaker:DeleteSimulationApplication
	robomaker:DeleteWorldTemplate
	robomaker:DeregisterRobot
	robomaker:DescribeDeploymentJob
	robomaker:DescribeFleet
	robomaker:DescribeRobot
	robomaker:DescribeRobotApplication
	robomaker:DescribeSimulationApplication
	robomaker:DescribeSimulationJob
	robomaker:DescribeSimulationJobBatch
	robomaker:DescribeWorld
	robomaker:DescribeWorldExportJob
	robomaker:DescribeWorldGenerationJob
	robomaker:DescribeWorldTemplate
	robomaker:GetWorldTemplateBody
	robomaker:ListDeploymentJobs
	robomaker:ListFleets
	robomaker:ListRobotApplications
	robomaker:ListRobots
	robomaker:ListSimulationApplications
	robomaker:ListSimulationJobBatches

Service prefix	Actions
	robomaker:ListSimulationJobs
	robomaker:ListWorldExportJobs
	robomaker:ListWorldGenerationJobs
	robomaker:ListWorldTemplates
	robomaker:ListWorlds
	robomaker:RegisterRobot
	robomaker:RestartSimulationJob
	robomaker:StartSimulationJobBatch
	robomaker:SyncDeploymentJob
	robomaker:UpdateRobotApplication
	robomaker:UpdateSimulationApplication
	robomaker:UpdateWorldTemplate

Service prefix	Actions
rolesanywhere	rolesanywhere:CreateProfile rolesanywhere:CreateTrustAnchor rolesanywhere:DeleteAttributeMapping rolesanywhere:DeleteCrl rolesanywhere:DeleteProfile rolesanywhere:DeleteTrustAnchor rolesanywhere:DisableCrl rolesanywhere:DisableProfile rolesanywhere:DisableTrustAnchor rolesanywhere:EnableCrl rolesanywhere:EnableProfile rolesanywhere:EnableTrustAnchor rolesanywhere:GetCrl rolesanywhere:GetProfile rolesanywhere:GetSubject rolesanywhere:GetTrustAnchor rolesanywhere:ImportCrl rolesanywhere:ListCrls rolesanywhere:ListProfiles rolesanywhere:ListSubjects rolesanywhere:ListTrustAnchors

Service prefix	Actions
	rolesanywhere:PutAttributeMapping rolesanywhere:PutNotificationSettings rolesanywhere:ResetNotificationSettings rolesanywhere:UpdateCrl rolesanywhere:UpdateProfile rolesanywhere:UpdateTrustAnchor

Service prefix	Actions
route53	route53:ActivateKeySigningKey
	route53:AssociateVPCWithHostedZone
	route53:ChangeCidrCollection
	route53:ChangeResourceRecordSets
	route53:CreateCidrCollection
	route53:CreateHealthCheck
	route53:CreateHostedZone
	route53:CreateKeySigningKey
	route53:CreateQueryLoggingConfig
	route53:CreateReusableDelegationSet
	route53:CreateTrafficPolicy
	route53:CreateTrafficPolicyInstance
	route53:CreateTrafficPolicyVersion
	route53:CreateVPCAssociationAuthorization
	route53:DeactivateKeySigningKey
	route53:DeleteCidrCollection
	route53:DeleteHealthCheck
	route53:DeleteHostedZone
	route53:DeleteKeySigningKey
	route53:DeleteQueryLoggingConfig
	route53:DeleteReusableDelegationSet

Service prefix	Actions
	route53:DeleteTrafficPolicy
	route53:DeleteTrafficPolicyInstance
	route53:DeleteVPCAssociationAuthorization
	route53:DisableHostedZoneDNSSEC
	route53:DisassociateVPCFromHostedZone
	route53:EnableHostedZoneDNSSEC
	route53:GetAccountLimit
	route53:GetChange
	route53:GetCheckerIpRanges
	route53:GetDNSSEC
	route53:GetGeoLocation
	route53:GetHealthCheck
	route53:GetHealthCheckCount
	route53:GetHealthCheckLastFailureReason
	route53:GetHealthCheckStatus
	route53:GetHostedZone
	route53:GetHostedZoneCount
	route53:GetHostedZoneLimit
	route53:GetQueryLoggingConfig
	route53:GetReusableDelegationSet
	route53:GetReusableDelegationSetLimit

Service prefix	Actions
	route53:GetTrafficPolicy
	route53:GetTrafficPolicyInstance
	route53:GetTrafficPolicyInstanceCount
	route53:ListCidrBlocks
	route53:ListCidrCollections
	route53:ListCidrLocations
	route53:ListGeoLocations
	route53:ListHealthChecks
	route53:ListHostedZones
	route53:ListHostedZonesByName
	route53:ListHostedZonesByVPC
	route53:ListQueryLoggingConfigs
	route53:ListResourceRecordSets
	route53:ListReusableDelegationSets
	route53:ListTrafficPolicies
	route53:ListTrafficPolicyInstances
	route53:ListTrafficPolicyInstancesByHostedZone
	route53:ListTrafficPolicyInstancesByPolicy
	route53:ListTrafficPolicyVersions
	route53:ListVPCAssociationAuthorizations
	route53:TestDNSAnswer

Service prefix	Actions
	route53:UpdateHealthCheck route53:UpdateHostedZoneComment route53:UpdateTrafficPolicyComment route53:UpdateTrafficPolicyInstance

Service prefix	Actions
route53-recovery-control-config	route53-recovery-control-config:CreateCluster route53-recovery-control-config:CreateControlPanel route53-recovery-control-config:CreateRoutingControl route53-recovery-control-config:CreateSafetyRule route53-recovery-control-config:DeleteCluster route53-recovery-control-config:DeleteControlPanel route53-recovery-control-config:DeleteRoutingControl route53-recovery-control-config:DeleteSafetyRule route53-recovery-control-config:DescribeCluster route53-recovery-control-config:DescribeControlPanel route53-recovery-control-config:DescribeRoutingControl route53-recovery-control-config:DescribeSafetyRule route53-recovery-control-config:GetResourcePolicy route53-recovery-control-config:ListAssociatedRoute53HealthChecks route53-recovery-control-config:ListClusters route53-recovery-control-config:ListControlPanels route53-recovery-control-config:ListRoutingControls route53-recovery-control-config:ListSafetyRules route53-recovery-control-config:UpdateCluster route53-recovery-control-config:UpdateControlPanel

Service prefix	Actions
	route53-recovery-control-config:UpdateRoutingControl route53-recovery-control-config:UpdateSafetyRule

Service prefix	Actions
route53-recovery-readiness	route53-recovery-readiness:CreateCell route53-recovery-readiness:CreateCrossAccountAuthorization route53-recovery-readiness:CreateReadinessCheck route53-recovery-readiness:CreateRecoveryGroup route53-recovery-readiness:CreateResourceSet route53-recovery-readiness:DeleteCell route53-recovery-readiness:DeleteCrossAccountAuthorization route53-recovery-readiness:DeleteReadinessCheck route53-recovery-readiness:DeleteRecoveryGroup route53-recovery-readiness:DeleteResourceSet route53-recovery-readiness:GetArchitectureRecommendations route53-recovery-readiness:GetCell route53-recovery-readiness:GetCellReadinessSummary route53-recovery-readiness:GetReadinessCheck route53-recovery-readiness:GetReadinessCheckResourceStatus route53-recovery-readiness:GetReadinessCheckStatus route53-recovery-readiness:GetRecoveryGroup route53-recovery-readiness:GetRecoveryGroupReadinessSummary route53-recovery-readiness:GetResourceSet route53-recovery-readiness:ListCells route53-recovery-readiness:ListCrossAccountAuthorizations

Service prefix	Actions
	route53-recovery-readiness:ListReadinessChecks
	route53-recovery-readiness:ListRecoveryGroups
	route53-recovery-readiness:ListResourceSets
	route53-recovery-readiness:ListRules
	route53-recovery-readiness:UpdateCell
	route53-recovery-readiness:UpdateReadinessCheck
	route53-recovery-readiness:UpdateRecoveryGroup
	route53-recovery-readiness:UpdateResourceSet

Service prefix	Actions
route53resolver	route53resolver:AssociateFirewallRuleGroup route53resolver:AssociateResolverEndpointIpAddress route53resolver:AssociateResolverQueryLogConfig route53resolver:AssociateResolverRule route53resolver:CreateFirewallDomainList route53resolver:CreateFirewallRule route53resolver:CreateFirewallRuleGroup route53resolver:CreateResolverEndpoint route53resolver:CreateResolverQueryLogConfig route53resolver:CreateResolverRule route53resolver:DeleteFirewallDomainList route53resolver:DeleteFirewallRule route53resolver:DeleteFirewallRuleGroup route53resolver:DeleteOutpostResolver route53resolver:DeleteResolverEndpoint route53resolver:DeleteResolverQueryLogConfig route53resolver:DeleteResolverRule route53resolver:DisassociateFirewallRuleGroup route53resolver:DisassociateResolverEndpointIpAddress route53resolver:DisassociateResolverQueryLogConfig route53resolver:DisassociateResolverRule

Service prefix	Actions
	route53resolver:GetFirewallConfig route53resolver:GetFirewallDomainList route53resolver:GetFirewallRuleGroup route53resolver:GetFirewallRuleGroupAssociation route53resolver:GetFirewallRuleGroupPolicy route53resolver:GetOutpostResolver route53resolver:GetResolverConfig route53resolver:GetResolverDnssecConfig route53resolver:GetResolverEndpoint route53resolver:GetResolverQueryLogConfig route53resolver:GetResolverQueryLogConfigAssociation route53resolver:GetResolverQueryLogConfigPolicy route53resolver:GetResolverRule route53resolver:GetResolverRuleAssociation route53resolver:GetResolverRulePolicy route53resolver:ImportFirewallDomains route53resolver:ListFirewallConfigs route53resolver:ListFirewallDomainLists route53resolver:ListFirewallDomains route53resolver:ListFirewallRuleGroupAssociations route53resolver:ListFirewallRuleGroups

Service prefix	Actions
	route53resolver:ListFirewallRuleTypes route53resolver:ListFirewallRules route53resolver:ListOutpostResolvers route53resolver:ListResolverConfigs route53resolver:ListResolverDnssecConfigs route53resolver:ListResolverEndpointIpAddresses route53resolver:ListResolverEndpoints route53resolver:ListResolverQueryLogConfigAssociations route53resolver:ListResolverQueryLogConfigs route53resolver:ListResolverRuleAssociations route53resolver:ListResolverRules route53resolver:PutFirewallRuleGroupPolicy route53resolver:PutResolverQueryLogConfigPolicy route53resolver:UpdateFirewallConfig route53resolver:UpdateFirewallDomains route53resolver:UpdateFirewallRule route53resolver:UpdateFirewallRuleGroupAssociation route53resolver:UpdateOutpostResolver route53resolver:UpdateResolverConfig route53resolver:UpdateResolverDnssecConfig route53resolver:UpdateResolverEndpoint

Service prefix	Actions
	route53resolver:UpdateResolverRule
rum	rum:BatchCreateRumMetricDefinitions rum:BatchDeleteRumMetricDefinitions rum:BatchGetRumMetricDefinitions rum:CreateAppMonitor rum:DeleteAppMonitor rum:DeleteResourcePolicy rum:DeleteRumMetricsDestination rum:GetAppMonitor rum:GetAppMonitorData rum:GetResourcePolicy rum:ListAppMonitors rum:ListRumMetricsDestinations rum:PutResourcePolicy rum:PutRumMetricsDestination rum:UpdateAppMonitor rum:UpdateRumMetricDefinition

Service prefix	Actions
s3	s3:AssociateAccessGrantsIdentityCenter s3:CreateAccessGrant s3:CreateAccessGrantsInstance s3:CreateAccessGrantsLocation s3:CreateAccessPoint s3:CreateAccessPointForObjectLambda s3:CreateBucket s3:CreateBucketMetadataTableConfiguration s3:CreateJob s3:CreateMultiRegionAccessPoint s3:DeleteAccessGrant s3:DeleteAccessGrantsInstance s3:DeleteAccessGrantsInstanceResourcePolicy s3:DeleteAccessGrantsLocation s3:DeleteAccessPoint s3:DeleteAccessPointForObjectLambda s3:DeleteAccessPointPolicy s3:DeleteAccessPointPolicyForObjectLambda s3:DeleteBucket s3:DeleteBucketMetadataTableConfiguration s3:DeleteBucketPolicy

Service prefix	Actions
	s3:DeleteBucketWebsite s3:DeleteMultiRegionAccessPoint s3:DeleteStorageLensConfiguration s3:DescribeJob s3:DescribeMultiRegionAccessPointOperation s3:DissociateAccessGrantsIdentityCenter s3:GetAccelerateConfiguration s3:GetAccessGrant s3:GetAccessGrantsInstance s3:GetAccessGrantsInstanceForPrefix s3:GetAccessGrantsInstanceResourcePolicy s3:GetAccessGrantsLocation s3:GetAccessPoint s3:GetAccessPointConfigurationForObjectLambda s3:GetAccessPointForObjectLambda s3:GetAccessPointPolicy s3:GetAccessPointPolicyForObjectLambda s3:GetAccessPointPolicyStatus s3:GetAccessPointPolicyStatusForObjectLambda s3:GetAccountPublicAccessBlock s3:GetAnalyticsConfiguration

Service prefix	Actions
	s3:GetBucketAbac s3:GetBucketAcl s3:GetBucketCORS s3:GetBucketLocation s3:GetBucketLogging s3:GetBucketNotification s3:GetBucketObjectLockConfiguration s3:GetBucketOwnershipControls s3:GetBucketPolicy s3:GetBucketPolicyStatus s3:GetBucketPublicAccessBlock s3:GetBucketRequestPayment s3:GetBucketVersioning s3:GetBucketWebsite s3:GetDataAccess s3:GetEncryptionConfiguration s3:GetIntelligentTieringConfiguration s3:GetInventoryConfiguration s3:GetLifecycleConfiguration s3:GetMetricsConfiguration s3:GetMultiRegionAccessPoint

Service prefix	Actions
	s3:GetMultiRegionAccessPointPolicy s3:GetMultiRegionAccessPointPolicyStatus s3:GetMultiRegionAccessPointRoutes s3:GetReplicationConfiguration s3:GetStorageLensConfiguration s3:GetStorageLensDashboard s3:ListAccessGrants s3:ListAccessGrantsInstances s3:ListAccessGrantsLocations s3:ListAccessPoints s3:ListAccessPointsForObjectLambda s3:ListAllMyBuckets s3:ListBucketMultipartUploads s3:ListCallerAccessGrants s3:ListJobs s3:ListMultiRegionAccessPoints s3:ListStorageLensConfigurations s3:PutAccelerateConfiguration s3:PutAccessGrantsInstanceResourcePolicy s3:PutAccessPointConfigurationForObjectLambda s3:PutAccessPointPolicy

Service prefix	Actions
	s3:PutAccessPointPolicyForObjectLambda s3:PutAccountPublicAccessBlock s3:PutAnalyticsConfiguration s3:PutBucketAbac s3:PutBucketAcl s3:PutBucketCORS s3:PutBucketLogging s3:PutBucketNotification s3:PutBucketObjectLockConfiguration s3:PutBucketOwnershipControls s3:PutBucketPolicy s3:PutBucketPublicAccessBlock s3:PutBucketRequestPayment s3:PutBucketVersioning s3:PutBucketWebsite s3:PutEncryptionConfiguration s3:PutIntelligentTieringConfiguration s3:PutInventoryConfiguration s3:PutLifecycleConfiguration s3:PutMetricsConfiguration s3:PutMultiRegionAccessPointPolicy

Service prefix	Actions
	s3:PutReplicationConfiguration s3:PutStorageLensConfiguration s3:SubmitMultiRegionAccessPointRoutes s3:UpdateAccessGrantsLocation s3:UpdateBucketMetadataAnnotationTableConfiguration s3:UpdateBucketMetadataJournalTableConfiguration s3:UpdateJobPriority s3:UpdateJobStatus
s3-outposts	s3-outposts:CreateEndpoint s3-outposts:DeleteEndpoint s3-outposts:ListEndpoints s3-outposts:ListOutpostsWithS3 s3-outposts:ListSharedEndpoints

Service prefix	Actions
sagemaker-geospatial	sagemaker-geospatial:DeleteEarthObservationJob sagemaker-geospatial:DeleteVectorEnrichmentJob sagemaker-geospatial:ExportEarthObservationJob sagemaker-geospatial:ExportVectorEnrichmentJob sagemaker-geospatial:GetEarthObservationJob sagemaker-geospatial:GetRasterDataCollection sagemaker-geospatial:GetTile sagemaker-geospatial:GetVectorEnrichmentJob sagemaker-geospatial:ListEarthObservationJobs sagemaker-geospatial:ListRasterDataCollections sagemaker-geospatial:ListVectorEnrichmentJobs sagemaker-geospatial:SearchRasterDataCollection sagemaker-geospatial:StartEarthObservationJob sagemaker-geospatial:StartVectorEnrichmentJob sagemaker-geospatial:StopEarthObservationJob sagemaker-geospatial:StopVectorEnrichmentJob

Service prefix	Actions
savingsplans	savingsplans:CreateSavingsPlan
	savingsplans:DeleteQueuedSavingsPlan
	savingsplans:DescribeSavingsPlanRates
	savingsplans:DescribeSavingsPlans
	savingsplans:DescribeSavingsPlansOfferingRates
	savingsplans:DescribeSavingsPlansOfferings
	savingsplans:ReturnSavingsPlan

Service prefix	Actions
schemas	schemas:CreateDiscoverer schemas:CreateRegistry schemas:CreateSchema schemas:DeleteDiscoverer schemas:DeleteRegistry schemas:DeleteResourcePolicy schemas:DeleteSchema schemas:DeleteSchemaVersion schemas:DescribeCodeBinding schemas:DescribeDiscoverer schemas:DescribeRegistry schemas:DescribeSchema schemas:ExportSchema schemas:GetCodeBindingSource schemas:GetDiscoveredSchema schemas:GetResourcePolicy schemas:ListDiscoverers schemas:ListRegistries schemas:ListSchemaVersions schemas:ListSchemas schemas:PutCodeBinding

Service prefix	Actions
	schemas:PutResourcePolicy schemas:SearchSchemas schemas:StartDiscoverer schemas:StopDiscoverer schemas:UpdateDiscoverer schemas:UpdateRegistry schemas:UpdateSchema
sdb	sdb:CreateDomain sdb:DeleteDomain sdb:DomainMetadata sdb:ListDomains

Service prefix	Actions
secretsmanager	secretsmanager:CancelRotateSecret secretsmanager:CreateSecret secretsmanager:DeleteResourcePolicy secretsmanager:DeleteSecret secretsmanager:DescribeSecret secretsmanager:GetRandomPassword secretsmanager:GetResourcePolicy secretsmanager:GetSecretValue secretsmanager:ListSecretVersionIds secretsmanager:ListSecrets secretsmanager:PutResourcePolicy secretsmanager:PutSecretValue secretsmanager:RemoveRegionsFromReplication secretsmanager:ReplicateSecretToRegions secretsmanager:RestoreSecret secretsmanager:RotateSecret secretsmanager:StopReplicationToReplica secretsmanager:UpdateSecret secretsmanager:ValidateResourcePolicy

Service prefix	Actions
securityhub	securityhub:AcceptAdministratorInvitation
	securityhub:AcceptInvitation
	securityhub:BatchDeleteAutomationRules
	securityhub:BatchDisableStandards
	securityhub:BatchEnableStandards
	securityhub:BatchGetAutomationRules
	securityhub:BatchGetConfigurationPolicyAssociations
	securityhub:BatchGetEnabledRegionsV2
	securityhub:BatchGetSecurityControls
	securityhub:BatchGetStandardsControlAssociations
	securityhub:BatchImportFindings
	securityhub:BatchUpdateAutomationRules
	securityhub:BatchUpdateFindings
	securityhub:BatchUpdateStandardsControlAssociations
	securityhub:ConnectorRegistrationsV2
	securityhub:CreateActionTarget
	securityhub:CreateAggregatorV2
	securityhub:CreateAutomationRule
	securityhub:CreateAutomationRuleV2
	securityhub:CreateConfigurationPolicy
	securityhub:CreateConnector

Service prefix	Actions
	securityhub:CreateConnectorV2
	securityhub:CreateFindingAggregator
	securityhub:CreateInsight
	securityhub:CreateMembers
	securityhub:CreateTicketV2
	securityhub:DeclineInvitations
	securityhub:DeleteActionTarget
	securityhub:DeleteAggregatorV2
	securityhub:DeleteAutomationRuleV2
	securityhub:DeleteConfigurationPolicy
	securityhub:DeleteConnector
	securityhub:DeleteConnectorV2
	securityhub:DeleteFindingAggregator
	securityhub:DeleteInsight
	securityhub:DeleteInvitations
	securityhub:DeleteMembers
	securityhub:DescribeActionTargets
	securityhub:DescribeHub
	securityhub:DescribeOrganizationConfiguration
	securityhub:DescribeProducts
	securityhub:DescribeSecurityHubV2

Service prefix	Actions
	securityhub:DescribeStandards
	securityhub:DisableImportFindingsForProduct
	securityhub:DisableOrganizationAdminAccount
	securityhub:DisableSecurityHub
	securityhub:DisableSecurityHubFeatureV2
	securityhub:DisableSecurityHubV2
	securityhub:DisassociateFromAdministratorAccount
	securityhub:DisassociateFromMasterAccount
	securityhub:DisassociateMembers
	securityhub:EnableImportFindingsForProduct
	securityhub:EnableOrganizationAdminAccount
	securityhub:EnableSecurityHub
	securityhub:EnableSecurityHubFeatureV2
	securityhub:GetAdministratorAccount
	securityhub:GetAggregatorV2
	securityhub:GetAutomationRuleV2
	securityhub:GetConfigurationPolicy
	securityhub:GetConfigurationPolicyAssociation
	securityhub:GetConnector
	securityhub:GetConnectorV2
	securityhub:GetCoverageStatisticsV2

Service prefix	Actions
	securityhub:GetEnabledStandards
	securityhub:GetFindingAggregator
	securityhub:GetFindingHistory
	securityhub:GetFindings
	securityhub:GetInsightResults
	securityhub:GetInsights
	securityhub:GetInvitationsCount
	securityhub:GetMasterAccount
	securityhub:GetMembers
	securityhub:GetSecurityControlDefinition
	securityhub:InviteMembers
	securityhub:ListAggregatorsV2
	securityhub:ListAutomationRules
	securityhub:ListAutomationRulesV2
	securityhub:ListConfigurationPolicies
	securityhub:ListConfigurationPolicyAssociations
	securityhub:ListConnectors
	securityhub:ListConnectorsV2
	securityhub:ListEnabledProductsForImport
	securityhub:ListFindingAggregators
	securityhub:ListInvitations

Service prefix	Actions
	securityhub:ListMembers
	securityhub:ListOrganizationAdminAccounts
	securityhub:ListSecurityControlDefinitions
	securityhub:ListStandardsControlAssociations
	securityhub:StartConfigurationPolicyAssociation
	securityhub:StartConfigurationPolicyDisassociation
	securityhub:UpdateActionTarget
	securityhub:UpdateAggregatorV2
	securityhub:UpdateAutomationRuleV2
	securityhub:UpdateConfigurationPolicy
	securityhub:UpdateConnector
	securityhub:UpdateConnectorV2
	securityhub:UpdateFindingAggregator
	securityhub:UpdateFindings
	securityhub:UpdateInsight
	securityhub:UpdateOrganizationConfiguration
	securityhub:UpdateSecurityControl
	securityhub:UpdateSecurityHubConfiguration

Service prefix	Actions
securitylake	securitylake:CreateAwsLogSource securitylake:CreateCustomLogSource securitylake:CreateDataLakeExceptionSubscription securitylake:CreateDataLakeOrganizationConfiguration securitylake:CreateSubscriber securitylake:CreateSubscriberNotification securitylake:DeleteAwsLogSource securitylake:DeleteCustomLogSource securitylake:DeleteDataLakeExceptionSubscription securitylake:DeleteDataLakeOrganizationConfiguration securitylake:DeleteSubscriber securitylake:DeleteSubscriberNotification securitylake:DeregisterDataLakeDelegatedAdministrator securitylake:GetDataLakeExceptionSubscription securitylake:GetDataLakeOrganizationConfiguration securitylake:GetDataLakeSources securitylake:GetSubscriber securitylake:ListDataLakes securitylake:ListLogSources securitylake:ListSubscribers securitylake:RegisterDataLakeDelegatedAdministrator

Service prefix	Actions
	securitylake:UpdateDataLakeExceptionSubscription securitylake:UpdateSubscriber securitylake:UpdateSubscriberNotification
serverlessrepo	serverlessrepo:CreateApplication serverlessrepo:CreateApplicationVersion serverlessrepo:CreateCloudFormationChangeSet serverlessrepo:CreateCloudFormationTemplate serverlessrepo:DeleteApplication serverlessrepo:GetApplication serverlessrepo:GetApplicationPolicy serverlessrepo:GetCloudFormationTemplate serverlessrepo:ListApplicationDependencies serverlessrepo:ListApplicationVersions serverlessrepo:ListApplications serverlessrepo:PutApplicationPolicy serverlessrepo:UnshareApplication serverlessrepo:UpdateApplication

Service prefix	Actions
servicecatalog	servicecatalog:AcceptPortfolioShare servicecatalog:AssociateBudgetWithResource servicecatalog:AssociatePrincipalWithPortfolio servicecatalog:AssociateProductWithPortfolio servicecatalog:AssociateServiceActionWithProvisioningArtifact servicecatalog:BatchAssociateServiceActionWithProvisioningArtifact servicecatalog:BatchDisassociateServiceActionFromProvisioningArtifact servicecatalog:CopyProduct servicecatalog:CreateAttributeGroup servicecatalog:CreateConstraint servicecatalog:CreatePortfolio servicecatalog:CreatePortfolioShare servicecatalog:CreateProduct servicecatalog:CreateProvisionedProductPlan servicecatalog:CreateProvisioningArtifact servicecatalog:CreateServiceAction servicecatalog:DeleteAttributeGroup servicecatalog:DeleteConstraint servicecatalog:DeletePortfolio servicecatalog:DeletePortfolioShare

Service prefix	Actions
	servicecatalog:DeleteProduct servicecatalog:DeleteProvisionedProductPlan servicecatalog:DeleteProvisioningArtifact servicecatalog:DeleteServiceAction servicecatalog:DescribeConstraint servicecatalog:DescribeCopyProductStatus servicecatalog:DescribePortfolio servicecatalog:DescribePortfolioShareStatus servicecatalog:DescribePortfolioShares servicecatalog:DescribeProduct servicecatalog:DescribeProductAsAdmin servicecatalog:DescribeProductView servicecatalog:DescribeProvisionedProduct servicecatalog:DescribeProvisionedProductPlan servicecatalog:DescribeProvisioningArtifact servicecatalog:DescribeProvisioningParameters servicecatalog:DescribeRecord servicecatalog:DescribeServiceAction servicecatalog:DescribeServiceActionExecutionParameters servicecatalog:DisableAWSOrganizationsAccess servicecatalog:DisassociateBudgetFromResource

Service prefix	Actions
	servicecatalog:DisassociatePrincipalFromPortfolio servicecatalog:DisassociateProductFromPortfolio servicecatalog:DisassociateServiceActionFromProvisioningArtifact servicecatalog:EnableAWSOrganizationsAccess servicecatalog:ExecuteProvisionedProductPlan servicecatalog:ExecuteProvisionedProductServiceAction servicecatalog:GetAWSOrganizationsAccessStatus servicecatalog:GetProvisionedProductOutputs servicecatalog:ImportAsProvisionedProduct servicecatalog:ListAcceptedPortfolioShares servicecatalog:ListAttributeGroups servicecatalog:ListBudgetsForResource servicecatalog:ListConstraintsForPortfolio servicecatalog:ListLaunchPaths servicecatalog:ListOrganizationPortfolioAccess servicecatalog:ListPortfolioAccess servicecatalog:ListPortfolios servicecatalog:ListPortfoliosForProduct servicecatalog:ListPrincipalsForPortfolio servicecatalog:ListProvisionedProductPlans servicecatalog:ListProvisioningArtifacts

Service prefix	Actions
	servicecatalog:ListProvisioningArtifactsForServiceAction servicecatalog:ListRecordHistory servicecatalog:ListServiceActions servicecatalog:ListServiceActionsForProvisioningArtifact servicecatalog:ListStackInstancesForProvisionedProduct servicecatalog:NotifyProvisionProductEngineWorkflowResult servicecatalog:NotifyTerminateProvisionedProductEngine WorkflowResult servicecatalog:NotifyUpdateProvisionedProductEngineWor kflowResult servicecatalog:ProvisionProduct servicecatalog:RejectPortfolioShare servicecatalog:ScanProvisionedProducts servicecatalog:SearchProducts servicecatalog:SearchProductsAsAdmin servicecatalog:SearchProvisionedProducts servicecatalog:TerminateProvisionedProduct servicecatalog:UpdateConstraint servicecatalog:UpdatePortfolio servicecatalog:UpdatePortfolioShare servicecatalog:UpdateProduct servicecatalog:UpdateProvisionedProduct

Service prefix	Actions
	servicecatalog:UpdateProvisionedProductProperties servicecatalog:UpdateProvisioningArtifact servicecatalog:UpdateServiceAction

Service prefix	Actions
servicediscovery	servicediscovery:CreateHttpNamespace
	servicediscovery:CreatePrivateDnsNamespace
	servicediscovery:CreatePublicDnsNamespace
	servicediscovery:CreateService
	servicediscovery:DeleteNamespace
	servicediscovery:DeleteService
	servicediscovery:DeleteServiceAttributes
	servicediscovery:DeregisterInstance
	servicediscovery:GetInstance
	servicediscovery:GetInstancesHealthStatus
	servicediscovery:GetNamespace
	servicediscovery:GetOperation
	servicediscovery:GetService
	servicediscovery:ListInstances
	servicediscovery:ListNamespaces
	servicediscovery:ListOperations
	servicediscovery:ListServices
	servicediscovery:RegisterInstance
	servicediscovery:UpdateHttpNamespace
	servicediscovery:UpdateInstanceCustomHealthStatus
	servicediscovery:UpdatePrivateDnsNamespace

Service prefix	Actions
	servicediscovery:UpdatePublicDnsNamespace servicediscovery:UpdateService servicediscovery:UpdateServiceAttributes

Service prefix	Actions
servicequotas	servicequotas:AssociateServiceQuotaTemplate servicequotas:CreateSupportCase servicequotas:DeleteServiceQuotaIncreaseRequestFromTemplate servicequotas:DisassociateServiceQuotaTemplate servicequotas:GetAWSDefaultServiceQuota servicequotas:GetAssociationForServiceQuotaTemplate servicequotas:GetAutoManagementConfiguration servicequotas:GetQuotaUtilizationReport servicequotas:GetRequestedServiceQuotaChange servicequotas:GetServiceQuota servicequotas:GetServiceQuotaIncreaseRequestFromTemplate servicequotas:ListAWSDefaultServiceQuotas servicequotas:ListRequestedServiceQuotaChangeHistory servicequotas:ListRequestedServiceQuotaChangeHistoryByQuota servicequotas:ListServiceQuotaIncreaseRequestsInTemplate servicequotas:ListServiceQuotas servicequotas:ListServices servicequotas:PutServiceQuotaIncreaseRequestIntoTemplate servicequotas:RequestServiceQuotaIncrease servicequotas:StartAutoManagement servicequotas:StartQuotaUtilizationReport

Service prefix	Actions
	servicequotas:StopAutoManagement servicequotas:UpdateAutoManagement

Service prefix	Actions
ses	ses:BatchGetMetricData
	ses:CloneReceiptRuleSet
	ses:CreateAddonInstance
	ses:CreateAddonSubscription
	ses:CreateAddressList
	ses:CreateAddressListImportJob
	ses:CreateArchive
	ses:CreateConfigurationSet
	ses:CreateConfigurationSetEventDestination
	ses:CreateConfigurationSetTrackingOptions
	ses:CreateContact
	ses:CreateContactList
	ses:CreateCustomVerificationEmailTemplate
	ses:CreateDedicatedIpPool
	ses:CreateDeliverabilityTestReport
	ses:CreateEmailIdentity
	ses:CreateEmailIdentityPolicy
	ses:CreateEmailTemplate
	ses:CreateImportJob
	ses:CreateIngressPoint
	ses:CreateMultiRegionEndpoint

Service prefix	Actions
	ses:CreateReceiptFilter
	ses:CreateReceiptRule
	ses:CreateReceiptRuleSet
	ses:CreateRelay
	ses:CreateRuleSet
	ses:CreateTemplate
	ses:CreateTenant
	ses:CreateTenantResourceAssociation
	ses:CreateTrafficPolicy
	ses:DeleteAddonInstance
	ses:DeleteAddonSubscription
	ses:DeleteAddressList
	ses:DeleteArchive
	ses:DeleteConfigurationSet
	ses:DeleteConfigurationSetEventDestination
	ses:DeleteConfigurationSetTrackingOptions
	ses:DeleteContact
	ses:DeleteContactList
	ses:DeleteCustomVerificationEmailTemplate
	ses:DeleteDedicatedIpPool
	ses:DeleteEmailIdentity

Service prefix	Actions
	ses:DeleteEmailIdentityPolicy
	ses:DeleteEmailTemplate
	ses:DeleteIdentity
	ses:DeleteIdentityPolicy
	ses:DeleteIngressPoint
	ses:DeleteMultiRegionEndpoint
	ses:DeleteReceiptFilter
	ses:DeleteReceiptRule
	ses:DeleteReceiptRuleSet
	ses:DeleteRelay
	ses:DeleteRuleSet
	ses:DeleteSuppressedDestination
	ses:DeleteTemplate
	ses:DeleteTenant
	ses:DeleteTenantResourceAssociation
	ses:DeleteTrafficPolicy
	ses:DeleteVerifiedEmailAddress
	ses:DeregisterMemberFromAddressList
	ses:DescribeActiveReceiptRuleSet
	ses:DescribeConfigurationSet
	ses:DescribeReceiptRule

Service prefix	Actions
	ses:DescribeReceiptRuleSet
	ses:GetAccount
	ses:GetAccountSendingEnabled
	ses:GetAddonInstance
	ses:GetAddonSubscription
	ses:GetAddressList
	ses:GetArchive
	ses:GetArchiveExport
	ses:GetArchiveMessage
	ses:GetArchiveMessageContent
	ses:GetArchiveSearch
	ses:GetArchiveSearchResults
	ses:GetBlacklistReports
	ses:GetConfigurationSet
	ses:GetConfigurationSetEventDestinations
	ses:GetContact
	ses:GetContactList
	ses:GetCustomVerificationEmailTemplate
	ses:GetDedicatedIp
	ses:GetDedicatedIpPool
	ses:GetDedicatedIps

Service prefix	Actions
	ses:GetDeliverabilityDashboardOptions
	ses:GetDeliverabilityTestReport
	ses:GetDomainDeliverabilityCampaign
	ses:GetDomainStatisticsReport
	ses:GetEmailAddressInsights
	ses:GetEmailIdentity
	ses:GetEmailIdentityPolicies
	ses:GetEmailTemplate
	ses:GetIdentityDkimAttributes
	ses:GetIdentityMailFromDomainAttributes
	ses:GetIdentityNotificationAttributes
	ses:GetIdentityPolicies
	ses:GetIdentityVerificationAttributes
	ses:GetImportJob
	ses:GetIngressPoint
	ses:GetMemberOfAddressList
	ses:GetMessageInsights
	ses:GetMultiRegionEndpoint
	ses:GetRelay
	ses:GetRuleSet
	ses:GetSendQuota

Service prefix	Actions
	ses:GetSendStatistics
	ses:GetSuppressedDestination
	ses:GetTemplate
	ses:GetTenant
	ses:GetTrafficPolicy
	ses:ListAddonInstances
	ses:ListAddonSubscriptions
	ses:ListAddressListImportJobs
	ses:ListAddressLists
	ses:ListArchiveExports
	ses:ListArchiveSearches
	ses:ListArchives
	ses:ListConfigurationSets
	ses:ListContactLists
	ses:ListContacts
	ses:ListCustomVerificationEmailTemplates
	ses:ListDedicatedIpPools
	ses:ListDeliverabilityTestReports
	ses:ListDomainDeliverabilityCampaigns
	ses:ListEmailIdentities
	ses:ListEmailTemplates

Service prefix	Actions
	ses:ListExportJobs
	ses:ListIdentities
	ses:ListIdentityPolicies
	ses:ListImportJobs
	ses:ListIngressPoints
	ses:ListMembersOfAddressList
	ses:ListMultiRegionEndpoints
	ses:ListReceiptFilters
	ses:ListReceiptRuleSets
	ses:ListRecommendations
	ses:ListRelays
	ses:ListReputationEntities
	ses:ListResourceTenants
	ses:ListRuleSets
	ses:ListSuppressedDestinations
	ses:ListTemplates
	ses:ListTenantResources
	ses:ListTenants
	ses:ListTrafficPolicies
	ses:ListVerifiedEmailAddresses
	ses:PutAccountDedicatedIpWarmupAttributes

Service prefix	Actions
	ses:PutAccountDetails
	ses:PutAccountSendingAttributes
	ses:PutAccountSuppressionAttributes
	ses:PutAccountVdmAttributes
	ses:PutConfigurationSetArchivingOptions
	ses:PutConfigurationSetDeliveryOptions
	ses:PutConfigurationSetReputationOptions
	ses:PutConfigurationSetSendingOptions
	ses:PutConfigurationSetSuppressionOptions
	ses:PutConfigurationSetTrackingOptions
	ses:PutConfigurationSetVdmOptions
	ses:PutDedicatedIpInPool
	ses:PutDedicatedIpPoolScalingAttributes
	ses:PutDedicatedIpWarmupAttributes
	ses:PutDeliverabilityDashboardOption
	ses:PutEmailIdentityConfigurationSetAttributes
	ses:PutEmailIdentityDkimAttributes
	ses:PutEmailIdentityDkimSigningAttributes
	ses:PutEmailIdentityFeedbackAttributes
	ses:PutEmailIdentityMailFromAttributes
	ses:PutIdentityPolicy

Service prefix	Actions
	ses:PutSuppressedDestination
	ses:PutTenantSuppressionAttributes
	ses:RegisterMemberToAddressList
	ses:ReorderReceiptRuleSet
	ses:SendBounce
	ses:SendCustomVerificationEmail
	ses:SetActiveReceiptRuleSet
	ses:SetIdentityDkimEnabled
	ses:SetIdentityFeedbackForwardingEnabled
	ses:SetIdentityHeadersInNotificationsEnabled
	ses:SetIdentityMailFromDomain
	ses:SetIdentityNotificationTopic
	ses:SetReceiptRulePosition
	ses:StartArchiveExport
	ses:StartArchiveSearch
	ses:StopArchiveExport
	ses:StopArchiveSearch
	ses:TestRenderEmailTemplate
	ses:TestRenderTemplate
	ses:UpdateAccountSendingEnabled
	ses:UpdateArchive

Service prefix	Actions
	ses:UpdateConfigurationSetEventDestination
	ses:UpdateConfigurationSetReputationMetricsEnabled
	ses:UpdateConfigurationSetSendingEnabled
	ses:UpdateConfigurationSetTrackingOptions
	ses:UpdateContact
	ses:UpdateContactList
	ses:UpdateCustomVerificationEmailTemplate
	ses:UpdateEmailIdentityPolicy
	ses:UpdateEmailTemplate
	ses:UpdateIngressPoint
	ses:UpdateReceiptRule
	ses:UpdateRelay
	ses:UpdateRuleSet
	ses:UpdateTemplate
	ses:UpdateTrafficPolicy
	ses:VerifyDomainDkim
	ses:VerifyDomainIdentity
	ses:VerifyEmailAddress
	ses:VerifyEmailIdentity

Service prefix	Actions
shield	shield:AssociateDRTLogBucket shield:AssociateHealthCheck shield:AssociateProactiveEngagementDetails shield:CreateProtection shield:CreateProtectionGroup shield:CreateSubscription shield:DeleteProtection shield:DeleteProtectionGroup shield:DeleteSubscription shield:DescribeAttack shield:DescribeAttackStatistics shield:DescribeDRTAccess shield:DescribeEmergencyContactSettings shield:DescribeProtection shield:DescribeProtectionGroup shield:DescribeSubscription shield:DisableApplicationLayerAutomaticResponse shield:DisableProactiveEngagement shield:DisassociateDRTLogBucket shield:DisassociateDRTRole shield:DisassociateHealthCheck

Service prefix	Actions
	shield:EnableApplicationLayerAutomaticResponse shield:EnableProactiveEngagement shield:GetSubscriptionState shield:ListAttacks shield:ListProtectionGroups shield:ListProtections shield:ListResourcesInProtectionGroup shield:UpdateApplicationLayerAutomaticResponse shield:UpdateEmergencyContactSettings shield:UpdateProtectionGroup shield:UpdateSubscription

Service prefix	Actions
signer	signer:AddProfilePermission signer:CancelSigningProfile signer:DescribeSigningJob signer:GetSigningPlatform signer:GetSigningProfile signer:ListProfilePermissions signer:ListSigningJobs signer:ListSigningPlatforms signer:ListSigningProfiles signer:PutSigningProfile signer:RemoveProfilePermission signer:RevokeSignature signer:RevokeSigningProfile signer:SignPayload signer:StartSigningJob

Service prefix	Actions
simspaceweaver	simspaceweaver:CreateSnapshot simspaceweaver:DeleteApp simspaceweaver:DeleteSimulation simspaceweaver:DescribeApp simspaceweaver:DescribeSimulation simspaceweaver:ListApps simspaceweaver:ListSimulations simspaceweaver:StartApp simspaceweaver:StartClock simspaceweaver:StartSimulation simspaceweaver:StopApp simspaceweaver:StopClock simspaceweaver:StopSimulation

Service prefix	Actions
sms	sms:CreateApp
	sms:CreateReplicationJob
	sms:DeleteApp
	sms:DeleteAppLaunchConfiguration
	sms:DeleteAppReplicationConfiguration
	sms:DeleteAppValidationConfiguration
	sms:DeleteReplicationJob
	sms:DeleteServerCatalog
	sms:DisassociateConnector
	sms:GenerateChangeSet
	sms:GenerateTemplate
	sms:GetApp
	sms:GetAppLaunchConfiguration
	sms:GetAppReplicationConfiguration
	sms:GetAppValidationConfiguration
	sms:GetAppValidationOutput
	sms:GetConnectors
	sms:GetReplicationJobs
	sms:GetReplicationRuns
	sms:GetServers
	sms:ImportAppCatalog

Service prefix	Actions
	sms:ImportServerCatalog
	sms:LaunchApp
	sms:ListApps
	sms:NotifyAppValidationOutput
	sms:PutAppLaunchConfiguration
	sms:PutAppReplicationConfiguration
	sms:PutAppValidationConfiguration
	sms:StartAppReplication
	sms:StartOnDemandAppReplication
	sms:StartOnDemandReplicationRun
	sms:StopAppReplication
	sms:TerminateApp
	sms:UpdateApp
	sms:UpdateReplicationJob

Service prefix	Actions
sms-voice	sms-voice:AssociateProtectConfiguration sms-voice:CreateConfigurationSet sms-voice:CreateConfigurationSetEventDestination sms-voice:CreateEventDestination sms-voice:CreateOptOutList sms-voice:CreatePool sms-voice:CreateProtectConfiguration sms-voice:CreateRegistration sms-voice:CreateRegistrationAssociation sms-voice:CreateRegistrationAttachment sms-voice:CreateRegistrationVersion sms-voice:CreateVerifiedDestinationNumber sms-voice:DeleteAccountDefaultProtectConfiguration sms-voice:DeleteConfigurationSet sms-voice:DeleteConfigurationSetEventDestination sms-voice:DeleteDefaultMessageType sms-voice:DeleteDefaultSenderId sms-voice:DeleteEventDestination sms-voice:DeleteKeyword sms-voice:DeleteMediaMessageSpendLimitOverride sms-voice:DeleteOptOutList

Service prefix	Actions
	sms-voice:DeleteOptedOutNumber sms-voice:DeletePool sms-voice:DeleteProtectConfiguration sms-voice:DeleteProtectConfigurationRuleSetNumberOverride sms-voice:DeleteRegistration sms-voice:DeleteRegistrationAttachment sms-voice:DeleteResourcePolicy sms-voice:DeleteTextMessageSpendLimitOverride sms-voice:DeleteVerifiedDestinationNumber sms-voice:DeleteVoiceMessageSpendLimitOverride sms-voice:DescribeAccountAttributes sms-voice:DescribeAccountLimits sms-voice:DescribeConfigurationSets sms-voice:DescribeKeywords sms-voice:DescribeOptOutLists sms-voice:DescribeOptedOutNumbers sms-voice:DescribePhoneNumbers sms-voice:DescribePools sms-voice:DescribeProtectConfigurations sms-voice:DescribeRegistrationAttachments sms-voice:DescribeRegistrationFieldDefinitions

Service prefix	Actions
	sms-voice:DescribeRegistrationFieldValues sms-voice:DescribeRegistrationSectionDefinitions sms-voice:DescribeRegistrationTypeDefinitions sms-voice:DescribeRegistrationVersions sms-voice:DescribeRegistrations sms-voice:DescribeSenderIds sms-voice:DescribeSpendLimits sms-voice:DescribeVerifiedDestinationNumbers sms-voice:DisassociateOriginationIdentity sms-voice:DisassociateProtectConfiguration sms-voice:DiscardRegistrationVersion sms-voice:GetConfigurationSetEventDestinations sms-voice:GetProtectConfigurationCountryRuleSet sms-voice:GetResourcePolicy sms-voice:ListConfigurationSets sms-voice:ListPoolOriginationIdentities sms-voice:ListProtectConfigurationRuleSetNumberOverrides sms-voice:ListRegistrationAssociations sms-voice:PutKeyword sms-voice:PutOptedOutNumber sms-voice:PutProtectConfigurationRuleSetNumberOverride

Service prefix	Actions
	sms-voice:PutResourcePolicy sms-voice:ReleasePhoneNumber sms-voice:ReleaseSenderId sms-voice:RequestPhoneNumber sms-voice:RequestSenderId sms-voice:SendDestinationNumberVerificationCode sms-voice:SetAccountDefaultProtectConfiguration sms-voice:SetDefaultMessageFeedbackEnabled sms-voice:SetDefaultMessageType sms-voice:SetDefaultSenderId sms-voice:SetMediaMessageSpendLimitOverride sms-voice:SetTextMessageSpendLimitOverride sms-voice:SetVoiceMessageSpendLimitOverride sms-voice:SubmitRegistrationVersion sms-voice:UpdateConfigurationSetEventDestination sms-voice:UpdateEventDestination sms-voice:UpdatePhoneNumber sms-voice:UpdatePool sms-voice:UpdateProtectConfiguration sms-voice:UpdateProtectConfigurationCountryRuleSet sms-voice:UpdateSenderId

Service prefix	Actions
snowball	snowball:CancelCluster
	snowball:CancelJob
	snowball:CreateAddress
	snowball:CreateCluster
	snowball:CreateJob
	snowball:CreateLongTermPricing
	snowball:CreateReturnShippingLabel
	snowball:DescribeAddress
	snowball:DescribeAddresses
	snowball:DescribeCluster
	snowball:DescribeJob
	snowball:DescribeReturnShippingLabel
	snowball:GetJobManifest
	snowball:GetJobUnlockCode
	snowball:GetSnowballUsage
	snowball:GetSoftwareUpdates
	snowball:ListClusterJobs
	snowball:ListClusters
	snowball:ListCompatibleImages
	snowball:ListJobs
	snowball:ListLongTermPricing

Service prefix	Actions
	snowball:ListPickupLocations snowball:ListServiceVersions snowball:UpdateCluster snowball:UpdateJob snowball:UpdateJobShipmentState snowball:UpdateLongTermPricing
sqs	sqs:AddPermission sqs:CancelMessageMoveTask sqs:CreateQueue sqs:DeleteQueue sqs:PurgeQueue sqs:RemovePermission sqs:SetQueueAttributes

Service prefix	Actions
ssm	ssm:AssociateOpsItemRelatedItem
	ssm:CancelCommand
	ssm:CancelMaintenanceWindowExecution
	ssm:CreateActivation
	ssm:CreateAssociation
	ssm:CreateAssociationBatch
	ssm:CreateDocument
	ssm:CreateMaintenanceWindow
	ssm:CreateOpsItem
	ssm:CreateOpsMetadata
	ssm:CreatePatchBaseline
	ssm:CreateResourceDataSync
	ssm:DeleteActivation
	ssm:DeleteAssociation
	ssm:DeleteCloudConnector
	ssm:DeleteDocument
	ssm:DeleteInventory
	ssm:DeleteMaintenanceWindow
	ssm:DeleteOpsItem
	ssm:DeleteOpsMetadata
	ssm:DeleteParameter

Service prefix	Actions
	ssm:DeleteParameters
	ssm:DeletePatchBaseline
	ssm:DeleteResourceDataSync
	ssm:DeleteResourcePolicy
	ssm:DeregisterManagedInstance
	ssm:DeregisterPatchBaselineForPatchGroup
	ssm:DeregisterTargetFromMaintenanceWindow
	ssm:DeregisterTaskFromMaintenanceWindow
	ssm:DescribeActivations
	ssm:DescribeAssociation
	ssm:DescribeAssociationExecutionTargets
	ssm:DescribeAssociationExecutions
	ssm:DescribeAutomationExecutions
	ssm:DescribeAutomationStepExecutions
	ssm:DescribeAvailablePatches
	ssm:DescribeDocument
	ssm:DescribeDocumentParameters
	ssm:DescribeDocumentPermission
	ssm:DescribeEffectiveInstanceAssociations
	ssm:DescribeEffectivePatchesForPatchBaseline
	ssm:DescribeInstanceAssociationsStatus

Service prefix	Actions
	ssm:DescribeInstanceInformation
	ssm:DescribeInstancePatchStates
	ssm:DescribeInstancePatchStatesForPatchGroup
	ssm:DescribeInstancePatches
	ssm:DescribeInstanceProperties
	ssm:DescribeInventoryDeletions
	ssm:DescribeMaintenanceWindowExecutionTaskInvocations
	ssm:DescribeMaintenanceWindowExecutionTasks
	ssm:DescribeMaintenanceWindowExecutions
	ssm:DescribeMaintenanceWindowSchedule
	ssm:DescribeMaintenanceWindowTargets
	ssm:DescribeMaintenanceWindowTasks
	ssm:DescribeMaintenanceWindows
	ssm:DescribeMaintenanceWindowsForTarget
	ssm:DescribeOpsItems
	ssm:DescribeParameters
	ssm:DescribePatchBaselines
	ssm:DescribePatchGroupState
	ssm:DescribePatchGroups
	ssm:DescribePatchProperties
	ssm:DescribeSessions

Service prefix	Actions
	ssm:DisassociateOpsItemRelatedItem
	ssm:GetAccessToken
	ssm:GetAutomationExecution
	ssm:GetCalendarState
	ssm:GetCloudConnector
	ssm:GetCommandInvocation
	ssm:GetConnectionStatus
	ssm:GetDefaultPatchBaseline
	ssm:GetDeployablePatchSnapshotForInstance
	ssm:GetDocument
	ssm:GetExecutionPreview
	ssm:GetInventory
	ssm:GetInventorySchema
	ssm:GetMaintenanceWindow
	ssm:GetMaintenanceWindowExecution
	ssm:GetMaintenanceWindowExecutionTask
	ssm:GetMaintenanceWindowExecutionTaskInvocation
	ssm:GetMaintenanceWindowTask
	ssm:GetOpsItem
	ssm:GetOpsMetadata
	ssm:GetOpsSummary

Service prefix	Actions
	ssm:GetParameter
	ssm:GetParameterHistory
	ssm:GetParameters
	ssm:GetParametersByPath
	ssm:GetPatchBaseline
	ssm:GetPatchBaselineForPatchGroup
	ssm:GetResourcePolicies
	ssm:GetServiceSetting
	ssm:LabelParameterVersion
	ssm:ListAssociationVersions
	ssm:ListAssociations
	ssm:ListCloudConnectors
	ssm:ListCommandInvocations
	ssm:ListCommands
	ssm:ListComplianceItems
	ssm:ListComplianceSummaries
	ssm:ListDocumentMetadataHistory
	ssm:ListDocumentVersions
	ssm:ListDocuments
	ssm:ListInstanceAssociations
	ssm:ListInventoryEntries

Service prefix	Actions
	ssm:ListNodes
	ssm:ListNodesSummary
	ssm:ListOpsItemEvents
	ssm:ListOpsItemRelatedItems
	ssm:ListOpsMetadata
	ssm:ListResourceComplianceSummaries
	ssm:ListResourceDataSync
	ssm:ModifyDocumentPermission
	ssm:PutComplianceItems
	ssm:PutInventory
	ssm:PutParameter
	ssm:PutResourcePolicy
	ssm:RegisterDefaultPatchBaseline
	ssm:RegisterManagedInstance
	ssm:RegisterPatchBaselineForPatchGroup
	ssm:RegisterTargetWithMaintenanceWindow
	ssm:RegisterTaskWithMaintenanceWindow
	ssm:ResetServiceSetting
	ssm:ResumeSession
	ssm:SendAutomationSignal
	ssm:SendCommand

Service prefix	Actions
	ssm:StartAssociationsOnce
	ssm:StartAutomationExecution
	ssm:StartChangeRequestExecution
	ssm:StartSession
	ssm:StopAutomationExecution
	ssm:TerminateSession
	ssm:UnlabelParameterVersion
	ssm:UpdateAssociation
	ssm:UpdateAssociationStatus
	ssm:UpdateCloudConnector
	ssm:UpdateDocument
	ssm:UpdateDocumentDefaultVersion
	ssm:UpdateDocumentMetadata
	ssm:UpdateInstanceInformation
	ssm:UpdateMaintenanceWindow
	ssm:UpdateMaintenanceWindowTarget
	ssm:UpdateMaintenanceWindowTask
	ssm:UpdateManagedInstanceRole
	ssm:UpdateOpsItem
	ssm:UpdateOpsMetadata
	ssm:UpdatePatchBaseline

Service prefix	Actions
	ssm:UpdateResourceDataSync ssm:UpdateServiceSetting ssm:ValidateCloudConnector

Service prefix	Actions
ssm-incidents	ssm-incidents:BatchGetIncidentFindings ssm-incidents:CreateReplicationSet ssm-incidents:CreateResponsePlan ssm-incidents:CreateTimelineEvent ssm-incidents:DeleteIncidentRecord ssm-incidents:DeleteReplicationSet ssm-incidents:DeleteResourcePolicy ssm-incidents:DeleteResponsePlan ssm-incidents:DeleteTimelineEvent ssm-incidents:GetIncidentRecord ssm-incidents:GetReplicationSet ssm-incidents:GetResourcePolicies ssm-incidents:GetResponsePlan ssm-incidents:GetTimelineEvent ssm-incidents:ListIncidentFindings ssm-incidents:ListIncidentRecords ssm-incidents:ListRelatedItems ssm-incidents:ListReplicationSets ssm-incidents:ListResponsePlans ssm-incidents:ListTimelineEvents ssm-incidents:PutResourcePolicy

Service prefix	Actions
	ssm-incidents:StartIncident ssm-incidents:UpdateDeletionProtection ssm-incidents:UpdateIncidentRecord ssm-incidents:UpdateRelatedItems ssm-incidents:UpdateReplicationSet ssm-incidents:UpdateResponsePlan ssm-incidents:UpdateTimelineEvent

Service prefix	Actions
ssm-sap	ssm-sap:BackupDatabase
	ssm-sap:DeleteResourcePermission
	ssm-sap:DeregisterApplication
	ssm-sap:GetApplication
	ssm-sap:GetComponent
	ssm-sap:GetConfigurationCheckOperation
	ssm-sap:GetDatabase
	ssm-sap:GetOperation
	ssm-sap:GetResourcePermission
	ssm-sap:ListApplications
	ssm-sap:ListComponents
	ssm-sap:ListConfigurationCheckDefinitions
	ssm-sap:ListConfigurationCheckOperations
	ssm-sap:ListDatabases
	ssm-sap:ListOperationEvents
	ssm-sap:ListOperations
	ssm-sap:ListSubCheckResults
	ssm-sap:ListSubCheckRuleResults
	ssm-sap:PutResourcePermission
	ssm-sap:RegisterApplication
	ssm-sap:RestoreDatabase

Service prefix	Actions
	ssm-sap:StartApplication ssm-sap:StartApplicationRefresh ssm-sap:StartConfigurationChecks ssm-sap:StopApplication ssm-sap:UpdateApplicationSettings ssm-sap:UpdateHANABackupSettings

Service prefix	Actions
states	states:CreateActivity states:CreateStateMachine states:CreateStateMachineAlias states:DeleteActivity states:DeleteStateMachine states:DeleteStateMachineAlias states:DeleteStateMachineVersion states:DescribeActivity states:DescribeExecution states:DescribeMapRun states:DescribeStateMachine states:DescribeStateMachineAlias states:DescribeStateMachineForExecution states:GetExecutionHistory states:ListActivities states:ListExecutions states:ListMapRuns states:ListStateMachineAliases states:ListStateMachineVersions states:ListStateMachines states:SendTaskFailure

Service prefix	Actions
	states:SendTaskHeartbeat states:SendTaskSuccess states:StartExecution states:StopExecution states:UpdateMapRun states:UpdateStateMachine states:UpdateStateMachineAlias states:ValidateStateMachineDefinition
sts	sts:AssumeRole sts:AssumeRoleWithSAML sts:AssumeRoleWithWebIdentity sts:DecodeAuthorizationMessage sts:GetAccessKeyInfo sts:GetCallerIdentity sts:GetFederationToken sts:GetSessionToken sts:GetWebIdentityToken

Service prefix	Actions
swf	swf:DeleteActivityType swf:DeleteWorkflowType swf:DeprecateActivityType swf:DeprecateDomain swf:DeprecateWorkflowType swf:DescribeActivityType swf:DescribeDomain swf:DescribeWorkflowType swf:ListActivityTypes swf:ListDomains swf:ListWorkflowTypes swf:RegisterActivityType swf:RegisterDomain swf:RegisterWorkflowType swf:UndeprecateActivityType swf:UndeprecateDomain swf:UndeprecateWorkflowType

Service prefix	Actions
synthetics	synthetics:AssociateResource
	synthetics:CreateCanary
	synthetics:CreateGroup
	synthetics:DeleteCanary
	synthetics:DeleteGroup
	synthetics:DescribeCanaries
	synthetics:DescribeCanariesLastRun
	synthetics:DescribeRuntimeVersions
	synthetics:DisassociateResource
	synthetics:GetCanary
	synthetics:GetCanaryRuns
	synthetics:GetGroup
	synthetics:ListAssociatedGroups
	synthetics:ListGroupResources
	synthetics:ListGroups
	synthetics:StartCanary
	synthetics:StartCanaryDryRun
	synthetics:StopCanary
	synthetics:UpdateCanary

Service prefix	Actions
tag	tag:DescribeReportCreation tag:GetComplianceSummary tag:GetResources tag:StartReportCreation

Service prefix	Actions
textract	textract:AnalyzeDocument
	textract:AnalyzeExpense
	textract:AnalyzeID
	textract:CreateAdapter
	textract:CreateAdapterVersion
	textract:DeleteAdapter
	textract:DeleteAdapterVersion
	textract:DetectDocumentText
	textract:GetAdapter
	textract:GetAdapterVersion
	textract:GetDocumentAnalysis
	textract:GetDocumentTextDetection
	textract:GetExpenseAnalysis
	textract:GetLendingAnalysis
	textract:GetLendingAnalysisSummary
	textract:ListAdapterVersions
	textract:ListAdapters
	textract:StartDocumentAnalysis
	textract:StartDocumentTextDetection
	textract:StartExpenseAnalysis
	textract:StartLendingAnalysis

Service prefix	Actions
	textract:UpdateAdapter

Service prefix	Actions
timestream	timestream:CancelQuery timestream:CreateDatabase timestream:CreateScheduledQuery timestream:CreateTable timestream:DeleteDatabase timestream:DeleteScheduledQuery timestream:DeleteTable timestream:DescribeAccountSettings timestream:DescribeDatabase timestream:DescribeScheduledQuery timestream:DescribeTable timestream:ExecuteScheduledQuery timestream:ListBatchLoadTasks timestream:ListDatabases timestream:ListScheduledQueries timestream:ListTables timestream:PrepareQuery timestream:UpdateAccountSettings timestream:UpdateDatabase timestream:UpdateScheduledQuery timestream:UpdateTable

Service prefix	Actions
tnb	tnb:CancelSolNetworkOperation
	tnb:CreateSolFunctionPackage
	tnb:CreateSolNetworkInstance
	tnb:CreateSolNetworkPackage
	tnb:DeleteSolFunctionPackage
	tnb:DeleteSolNetworkInstance
	tnb:DeleteSolNetworkPackage
	tnb:GetSolFunctionInstance
	tnb:GetSolFunctionPackage
	tnb:GetSolFunctionPackageContent
	tnb:GetSolFunctionPackageDescriptor
	tnb:GetSolNetworkInstance
	tnb:GetSolNetworkOperation
	tnb:GetSolNetworkPackage
	tnb:GetSolNetworkPackageContent
	tnb:GetSolNetworkPackageDescriptor
	tnb:InstantiateSolNetworkInstance
	tnb:ListSolFunctionInstances
	tnb:ListSolFunctionPackages
	tnb:ListSolNetworkInstances
	tnb:ListSolNetworkOperations

Service prefix	Actions
	tnb:ListSolNetworkPackages tnb:PutSolFunctionPackageContent tnb:PutSolNetworkPackageContent tnb:TerminateSolNetworkInstance tnb:UpdateSolFunctionPackage tnb:UpdateSolNetworkInstance tnb:UpdateSolNetworkPackage tnb:ValidateSolFunctionPackageContent tnb:ValidateSolNetworkPackageContent

Service prefix	Actions
transcribe	transcribe:CreateCallAnalyticsCategory transcribe:CreateLanguageModel transcribe:CreateMedicalVocabulary transcribe:CreateVocabulary transcribe:CreateVocabularyFilter transcribe:DeleteCallAnalyticsCategory transcribe:DeleteCallAnalyticsJob transcribe:DeleteLanguageModel transcribe:DeleteMedicalScribeJob transcribe:DeleteMedicalTranscriptionJob transcribe:DeleteMedicalVocabulary transcribe:DeleteTranscriptionJob transcribe:DeleteVocabulary transcribe:DeleteVocabularyFilter transcribe:DescribeLanguageModel transcribe:GetCallAnalyticsCategory transcribe:GetCallAnalyticsJob transcribe:GetMedicalScribeJob transcribe:GetMedicalTranscriptionJob transcribe:GetMedicalVocabulary transcribe:GetTranscriptionJob

Service prefix	Actions
	transcribe:GetVocabulary transcribe:GetVocabularyFilter transcribe:ListCallAnalyticsCategories transcribe:ListCallAnalyticsJobs transcribe:ListLanguageModels transcribe:ListMedicalScribeJobs transcribe:ListMedicalTranscriptionJobs transcribe:ListMedicalVocabularies transcribe:ListTranscriptionJobs transcribe:ListVocabularies transcribe:ListVocabularyFilters transcribe:StartCallAnalyticsJob transcribe:StartCallAnalyticsStreamTranscription transcribe:StartCallAnalyticsStreamTranscriptionWebSocket transcribe:StartMedicalScribeJob transcribe:StartMedicalStreamTranscription transcribe:StartMedicalStreamTranscriptionWebSocket transcribe:StartMedicalTranscriptionJob transcribe:StartStreamTranscription transcribe:StartStreamTranscriptionWebSocket transcribe:StartTranscriptionJob

Service prefix	Actions
	transcribe:UpdateCallAnalyticsCategory transcribe:UpdateMedicalVocabulary transcribe:UpdateVocabulary transcribe:UpdateVocabularyFilter

Service prefix	Actions
transfer	transfer:CreateAccess
	transfer:CreateAgreement
	transfer:CreateConnector
	transfer:CreateProfile
	transfer:CreateServer
	transfer:CreateUser
	transfer:CreateWebApp
	transfer:CreateWorkflow
	transfer:DeleteAccess
	transfer:DeleteAgreement
	transfer:DeleteCertificate
	transfer:DeleteConnector
	transfer:DeleteHostKey
	transfer:DeleteProfile
	transfer:DeleteServer
	transfer:DeleteSshPublicKey
	transfer:DeleteUser
	transfer:DeleteWebApp
	transfer:DeleteWebAppCustomization
	transfer:DeleteWorkflow
	transfer:DescribeAccess

Service prefix	Actions
	transfer:DescribeAgreement
	transfer:DescribeCertificate
	transfer:DescribeConnector
	transfer:DescribeExecution
	transfer:DescribeHostKey
	transfer:DescribeProfile
	transfer:DescribeSecurityPolicy
	transfer:DescribeServer
	transfer:DescribeUser
	transfer:DescribeWebApp
	transfer:DescribeWebAppCustomization
	transfer:DescribeWorkflow
	transfer:ImportCertificate
	transfer:ImportHostKey
	transfer:ImportSshPublicKey
	transfer:ListAccesses
	transfer:ListCertificates
	transfer:ListConnectors
	transfer:ListExecutions
	transfer:ListFileTransferResults
	transfer:ListHostKeys

Service prefix	Actions
	transfer:ListProfiles
	transfer:ListSecurityPolicies
	transfer:ListServers
	transfer:ListUsers
	transfer:ListWebApps
	transfer:ListWorkflows
	transfer:SendWorkflowStepState
	transfer:StartDirectoryListing
	transfer:StartFileTransfer
	transfer:StartRemoteDelete
	transfer:StartRemoteMove
	transfer:StartServer
	transfer:StopServer
	transfer:TestConnection
	transfer:TestIdentityProvider
	transfer:UpdateAccess
	transfer:UpdateAgreement
	transfer:UpdateCertificate
	transfer:UpdateConnector
	transfer:UpdateHostKey
	transfer:UpdateProfile

Service prefix	Actions
	transfer:UpdateServer transfer:UpdateUser transfer:UpdateWebApp transfer:UpdateWebAppCustomization
translate	translate:CreateParallelData translate:DeleteParallelData translate:DeleteTerminology translate:DescribeTextTranslationJob translate:GetParallelData translate:GetTerminology translate:ImportTerminology translate:ListLanguages translate:ListParallelData translate:ListTerminologies translate:ListTextTranslationJobs translate:StartTextTranslationJob translate:StopTextTranslationJob translate:TranslateDocument translate:TranslateText translate:UpdateParallelData

Service prefix	Actions
voiceid	voiceid:AssociateFraudster
	voiceid:CreateDomain
	voiceid:CreateWatchlist
	voiceid:DeleteDomain
	voiceid:DeleteFraudster
	voiceid:DeleteSpeaker
	voiceid:DeleteWatchlist
	voiceid:DescribeDomain
	voiceid:DescribeFraudster
	voiceid:DescribeFraudsterRegistrationJob
	voiceid:DescribeSpeaker
	voiceid:DescribeSpeakerEnrollmentJob
	voiceid:DescribeWatchlist
	voiceid:DisassociateFraudster
	voiceid:EvaluateSession
	voiceid:ListDomains
	voiceid:ListFraudsterRegistrationJobs
	voiceid:ListFraudsters
	voiceid:ListSpeakerEnrollmentJobs
	voiceid:ListSpeakers
	voiceid:ListWatchlists

Service prefix	Actions
	voiceid:OptOutSpeaker voiceid:StartFraudsterRegistrationJob voiceid:StartSpeakerEnrollmentJob voiceid:UpdateDomain voiceid:UpdateWatchlist

Service prefix	Actions
vpc-lattice	vpc-lattice:CreateAccessLogSubscription vpc-lattice:CreateListener vpc-lattice:CreateResourceConfiguration vpc-lattice:CreateResourceGateway vpc-lattice:CreateRule vpc-lattice:CreateService vpc-lattice:CreateServiceNetwork vpc-lattice:CreateServiceNetworkResourceAssociation vpc-lattice:CreateServiceNetworkServiceAssociation vpc-lattice:CreateServiceNetworkVpcAssociation vpc-lattice:CreateTargetGroup vpc-lattice:DeleteAccessLogSubscription vpc-lattice:DeleteAuthPolicy vpc-lattice:DeleteDomainVerification vpc-lattice:DeleteListener vpc-lattice:DeleteResourceConfiguration vpc-lattice:DeleteResourceEndpointAssociation vpc-lattice:DeleteResourceGateway vpc-lattice:DeleteResourcePolicy vpc-lattice:DeleteRule vpc-lattice:DeleteService

Service prefix	Actions
	vpc-lattice:DeleteServiceNetwork vpc-lattice:DeleteServiceNetworkResourceAssociation vpc-lattice:DeleteServiceNetworkServiceAssociation vpc-lattice:DeleteServiceNetworkVpcAssociation vpc-lattice:DeleteTargetGroup vpc-lattice:DeregisterTargets vpc-lattice:GetAccessLogSubscription vpc-lattice:GetAuthPolicy vpc-lattice:GetDomainVerification vpc-lattice:GetListener vpc-lattice:GetResourceConfiguration vpc-lattice:GetResourceGateway vpc-lattice:GetResourcePolicy vpc-lattice:GetRule vpc-lattice:GetService vpc-lattice:GetServiceNetwork vpc-lattice:GetServiceNetworkResourceAssociation vpc-lattice:GetServiceNetworkServiceAssociation vpc-lattice:GetServiceNetworkVpcAssociation vpc-lattice:GetTargetGroup vpc-lattice:ListAccessLogSubscriptions

Service prefix	Actions
	vpc-lattice:ListDomainVerifications vpc-lattice:ListListeners vpc-lattice:ListResourceConfigurations vpc-lattice:ListResourceEndpointAssociations vpc-lattice:ListResourceGateways vpc-lattice:ListRules vpc-lattice:ListServiceNetworkResourceAssociations vpc-lattice:ListServiceNetworkServiceAssociations vpc-lattice:ListServiceNetworkVpcAssociations vpc-lattice:ListServiceNetworkVpcEndpointAssociations vpc-lattice:ListServiceNetworks vpc-lattice:ListServices vpc-lattice:ListTargetGroups vpc-lattice:ListTargets vpc-lattice:PutAuthPolicy vpc-lattice:PutResourcePolicy vpc-lattice:RegisterTargets vpc-lattice:StartDomainVerification vpc-lattice:UpdateAccessLogSubscription vpc-lattice:UpdateListener vpc-lattice:UpdateResourceConfiguration

Service prefix	Actions
	vpc-lattice:UpdateResourceGateway vpc-lattice:UpdateRule vpc-lattice:UpdateService vpc-lattice:UpdateServiceNetwork vpc-lattice:UpdateServiceNetworkVpcAssociation vpc-lattice:UpdateTargetGroup

Service prefix	Actions
wafv2	wafv2:AssociateWebACL
	wafv2:CheckCapacity
	wafv2:CreateAPIKey
	wafv2:CreateIPSet
	wafv2:CreateRegexPatternSet
	wafv2:CreateRuleGroup
	wafv2:CreateWebACL
	wafv2:DeleteAPIKey
	wafv2:DeleteFirewallManagerRuleGroups
	wafv2:DeleteIPSet
	wafv2:DeleteLoggingConfiguration
	wafv2:DeletePermissionPolicy
	wafv2:DeleteRegexPatternSet
	wafv2:DeleteRuleGroup
	wafv2:DeleteWebACL
	wafv2:DescribeAllManagedProducts
	wafv2:DescribeManagedProductsByVendor
	wafv2:DescribeManagedRuleGroup
	wafv2:DisassociateWebACL
	wafv2:GenerateMobileSdkReleaseUrl
	wafv2:GetDecryptedAPIKey

Service prefix	Actions
	wafv2:GetIPSet
	wafv2:GetLoggingConfiguration
	wafv2:GetManagedRuleSet
	wafv2:GetMobileSdkRelease
	wafv2:GetRateBasedStatementManagedKeys
	wafv2:GetRegexPatternSet
	wafv2:GetRevenueStatistics
	wafv2:GetRevenueStatisticsSummary
	wafv2:GetRevenueStatisticsTimeSeries
	wafv2:GetRuleGroup
	wafv2:GetSampledRequests
	wafv2:GetWebACLForResource
	wafv2:ListAPIKeys
	wafv2:ListAvailableManagedRuleGroupVersions
	wafv2:ListAvailableManagedRuleGroups
	wafv2:ListIPSets
	wafv2:ListLoggingConfigurations
	wafv2:ListManagedRuleSets
	wafv2:ListMobileSdkReleases
	wafv2:ListRegexPatternSets
	wafv2:ListResourcesForWebACL

Service prefix	Actions
	wafv2:ListRuleGroups
	wafv2:ListSettlementRecords
	wafv2:ListWebACLs
	wafv2:PutLoggingConfiguration
	wafv2:PutManagedRuleSetVersions
	wafv2:UpdateIPSet
	wafv2:UpdateManagedRuleSetVersionExpiryDate
	wafv2:UpdateRegexPatternSet
	wafv2:UpdateRuleGroup
	wafv2:UpdateWebACL

Service prefix	Actions
wellarchitected	wellarchitected:AssociateLenses wellarchitected:AssociateProfiles wellarchitected:CreateLensShare wellarchitected:CreateLensVersion wellarchitected:CreateMilestone wellarchitected:CreateProfile wellarchitected:CreateProfileShare wellarchitected:CreateReviewTemplate wellarchitected:CreateWorkload wellarchitected:CreateWorkloadShare wellarchitected:DeleteLens wellarchitected:DeleteLensShare wellarchitected:DeleteProfile wellarchitected:DeleteProfileShare wellarchitected:DeleteReviewTemplate wellarchitected:DeleteTemplateShare wellarchitected:DeleteWorkload wellarchitected:DeleteWorkloadShare wellarchitected:DisassociateLenses wellarchitected:DisassociateProfiles wellarchitected:ExportLens

Service prefix	Actions
	wellarchitected:GetAnswer wellarchitected:GetConsolidatedReport wellarchitected:GetGlobalSettings wellarchitected:GetLens wellarchitected:GetLensReview wellarchitected:GetLensReviewReport wellarchitected:GetLensVersionDifference wellarchitected:GetMilestone wellarchitected:GetProfile wellarchitected:GetProfileTemplate wellarchitected:GetReviewTemplate wellarchitected:GetReviewTemplateAnswer wellarchitected:GetReviewTemplateLensReview wellarchitected:GetWorkload wellarchitected:ImportLens wellarchitected:ListAnswers wellarchitected:ListCheckDetails wellarchitected:ListCheckSummaries wellarchitected:ListLensReviewImprovements wellarchitected:ListLensReviews wellarchitected:ListLensShares

Service prefix	Actions
	wellarchitected:ListLenses wellarchitected:ListMilestones wellarchitected:ListNotifications wellarchitected:ListProfileNotifications wellarchitected:ListProfileShares wellarchitected:ListProfiles wellarchitected:ListReviewTemplateAnswers wellarchitected:ListReviewTemplates wellarchitected:ListShareInvitations wellarchitected:ListTemplateShares wellarchitected:ListWorkloadShares wellarchitected:ListWorkloads wellarchitected:UpdateAnswer wellarchitected:UpdateGlobalSettings wellarchitected:UpdateIntegration wellarchitected:UpdateLensReview wellarchitected:UpdateProfile wellarchitected:UpdateReviewTemplate wellarchitected:UpdateReviewTemplateLensReview wellarchitected:UpdateShareInvitation wellarchitected:UpdateWorkload

Service prefix	Actions
	wellarchitected:UpdateWorkloadShare wellarchitected:UpgradeLensReview wellarchitected:UpgradeProfileVersion wellarchitected:UpgradeReviewTemplateLensReview

Service prefix	Actions
wisdom	wisdom:CreateAssistant
	wisdom:CreateAssistantAssociation
	wisdom:CreateContent
	wisdom:CreateKnowledgeBase
	wisdom:CreateQuickResponse
	wisdom:CreateSession
	wisdom:DeleteAssistant
	wisdom:DeleteAssistantAssociation
	wisdom:DeleteContent
	wisdom:DeleteImportJob
	wisdom:DeleteKnowledgeBase
	wisdom:DeleteQuickResponse
	wisdom:GetAssistant
	wisdom:GetAssistantAssociation
	wisdom:GetContent
	wisdom:GetContentAssociation
	wisdom:GetContentSummary
	wisdom:GetImportJob
	wisdom:GetKnowledgeBase
	wisdom:GetRecommendations
	wisdom:GetSession

Service prefix	Actions
	wisdom:ListAssistantAssociations
	wisdom:ListAssistants
	wisdom:ListContentAssociations
	wisdom:ListContents
	wisdom:ListImportJobs
	wisdom:ListKnowledgeBases
	wisdom:ListQuickResponses
	wisdom:NotifyRecommendationsReceived
	wisdom:QueryAssistant
	wisdom:RemoveKnowledgeBaseTemplateUri
	wisdom:SearchContent
	wisdom:SearchQuickResponses
	wisdom:SearchSessions
	wisdom:StartContentUpload
	wisdom:StartImportJob
	wisdom:UpdateContent
	wisdom:UpdateKnowledgeBaseTemplateUri
	wisdom:UpdateQuickResponse
	wisdom:UpdateSession

Service prefix	Actions
worklink	worklink:AssociateDomain worklink:AssociateWebsiteAuthorizationProvider worklink:AssociateWebsiteCertificateAuthority worklink:CreateFleet worklink:DeleteFleet worklink:DescribeAuditStreamConfiguration worklink:DescribeCompanyNetworkConfiguration worklink:DescribeDevice worklink:DescribeDevicePolicyConfiguration worklink:DescribeDomain worklink:DescribeFleetMetadata worklink:DescribeIdentityProviderConfiguration worklink:DescribeWebsiteCertificateAuthority worklink:DisassociateDomain worklink:DisassociateWebsiteAuthorizationProvider worklink:DisassociateWebsiteCertificateAuthority worklink:ListDevices worklink:ListDomains worklink:ListFleets worklink:ListWebsiteAuthorizationProviders worklink:ListWebsiteCertificateAuthorities

Service prefix	Actions
	worklink:RestoreDomainAccess worklink:RevokeDomainAccess worklink:SignOutUser worklink:UpdateAuditStreamConfiguration worklink:UpdateCompanyNetworkConfiguration worklink:UpdateDevicePolicyConfiguration worklink:UpdateDomainMetadata worklink:UpdateFleetMetadata worklink:UpdateIdentityProviderConfiguration

Service prefix	Actions
workspaces	workspaces:AcceptAccountLinkInvitation
	workspaces:AssociateConnectionAlias
	workspaces:AssociateIpGroups
	workspaces:AssociateWorkspaceApplication
	workspaces:CopyWorkspaceImage
	workspaces:CreateAccountLinkInvitation
	workspaces:CreateConnectClientAddIn
	workspaces:CreateConnectionAlias
	workspaces:CreateIpGroup
	workspaces:CreateStandbyWorkspaces
	workspaces:CreateUpdatedWorkspaceImage
	workspaces:CreateWorkspaceBundle
	workspaces:CreateWorkspaceImage
	workspaces:CreateWorkspaces
	workspaces:CreateWorkspacesPool
	workspaces:DeleteAccountLinkInvitation
	workspaces:DeleteClientBranding
	workspaces:DeleteConnectClientAddIn
	workspaces:DeleteConnectionAlias
	workspaces:DeleteIpGroup
	workspaces:DeleteWorkspaceBundle

Service prefix	Actions
	workspaces:DeleteWorkspaceImage
	workspaces:DeployWorkspaceApplications
	workspaces:DeregisterWorkspaceDirectory
	workspaces:DescribeAccount
	workspaces:DescribeAccountModifications
	workspaces:DescribeApplicationAssociations
	workspaces:DescribeApplications
	workspaces:DescribeBundleAssociations
	workspaces:DescribeClientBranding
	workspaces:DescribeClientProperties
	workspaces:DescribeConnectClientAddIns
	workspaces:DescribeConnectionAliasPermissions
	workspaces:DescribeConnectionAliases
	workspaces:DescribeCustomWorkspaceImageImport
	workspaces:DescribeImageAssociations
	workspaces:DescribeIpGroups
	workspaces:DescribeWorkspaceAssociations
	workspaces:DescribeWorkspaceBundles
	workspaces:DescribeWorkspaceDirectories
	workspaces:DescribeWorkspaceImagePermissions
	workspaces:DescribeWorkspaceSnapshots

Service prefix	Actions
	workspaces:DescribeWorkspaces
	workspaces:DescribeWorkspacesConnectionStatus
	workspaces:DescribeWorkspacesPoolSessions
	workspaces:DescribeWorkspacesPools
	workspaces:DisassociateConnectionAlias
	workspaces:DisassociateIpGroups
	workspaces:DisassociateWorkspaceApplication
	workspaces:GetAccountLink
	workspaces:ImportClientBranding
	workspaces:ImportWorkspaceImage
	workspaces:ListAccountLinks
	workspaces:ListAvailableManagementCidrRanges
	workspaces:MigrateWorkspace
	workspaces:ModifyAccount
	workspaces:ModifyCertificateBasedAuthProperties
	workspaces:ModifyClientProperties
	workspaces:ModifyEndpointEncryptionMode
	workspaces:ModifySamlProperties
	workspaces:ModifySelfservicePermissions
	workspaces:ModifyStreamingProperties
	workspaces:ModifyWorkspaceAccessProperties

Service prefix	Actions
	workspaces:ModifyWorkspaceCreationProperties
	workspaces:ModifyWorkspaceProperties
	workspaces:ModifyWorkspaceState
	workspaces:RebootWorkspaces
	workspaces:RebuildWorkspaces
	workspaces:RegisterWorkspaceDirectory
	workspaces:RejectAccountLinkInvitation
	workspaces:RestoreWorkspace
	workspaces:StartWorkspaces
	workspaces:StartWorkspacesPool
	workspaces:StopWorkspaces
	workspaces:StopWorkspacesPool
	workspaces:TerminateWorkspaces
	workspaces:TerminateWorkspacesPool
	workspaces:TerminateWorkspacesPoolSession
	workspaces:UpdateConnectClientAddIn
	workspaces:UpdateConnectionAliasPermission
	workspaces:UpdateWorkspaceBundle
	workspaces:UpdateWorkspaceImagePermission
	workspaces:UpdateWorkspacesPool

Service prefix	Actions
xray	xray:CreateGroup xray:CreateSamplingRule xray:DeleteGroup xray:DeleteResourcePolicy xray:DeleteSamplingRule xray:GetEncryptionConfig xray:GetGroup xray:GetGroups xray:GetInsight xray:GetInsightEvents xray:GetInsightImpactGraph xray:GetInsightSummaries xray:GetSamplingRules xray:ListResourcePolicies xray:PutEncryptionConfig xray:PutResourcePolicy xray:UpdateGroup xray:UpdateSamplingRule

IAM Access Analyzer quotas

IAM Access Analyzer has the following quotas:

Resource	Default quota	Maximum quota
Maximum account-level analyzers per analyzer type per AWS account per Region	1	1
Maximum organization-level external or unused access analyzers per analyzer type per AWS account per Region	5	20[1]
Maximum organization-level internal access analyzers per AWS organization per Region	1	1
Maximum archive rules per analyzer	100 Each archive rule can have up to 20 values per criterion.	1,000[1]
Maximum number of access previews per analyzer per hour	1,000	1,000
AWS CloudTrail log files processed per policy generations	100,000	100,000
Concurrent policy generations	1	1
Policy generation AWS CloudTrail data size	25 GB	25 GB
Policy generation AWS CloudTrail time range	90 days	90 days
Policy generations per day	Africa (Cape Town): 5 Asia Pacific (Hong Kong): 5	Africa (Cape Town): 5 Asia Pacific (Hong Kong): 5

Resource	Default quota	Maximum quota
	Asia Pacific (Jakarta): 5 Europe (Milan): 5 Middle East (Bahrain): 5 All other supported regions: 50 **Note** Canceled policy generation requests apply to the daily quota.	Asia Pacific (Jakarta): 5 Europe (Milan): 5 Middle East (Bahrain): 5 All other supported regions: 50

[1]Some quotas are customer-configurable using Service Quotas.

Troubleshoot IAM

Use the information here to help you diagnose and fix common issues when you work with AWS Identity and Access Management (IAM).

Issues

- I can't sign in to my AWS account
- I lost my access keys
- Policy variables aren't working
- Changes that I make are not always immediately visible
- I am not authorized to perform: iam:DeleteVirtualMFADevice
- How do I securely create IAM users?
- Additional resources
- Troubleshoot access denied error messages
- Troubleshoot access denied error messages with authorization ID (Preview)
- Troubleshoot issues with the root user
- Troubleshoot IAM policies
- Troubleshoot Passkeys and FIDO Security Keys
- Troubleshoot IAM roles
- Troubleshoot IAM and Amazon EC2
- Troubleshoot IAM and Amazon S3
- Troubleshoot SAML federation with IAM

I can't sign in to my AWS account

Verify that you have the correct credentials and that you are using the correct method to sign in. For more information, see Troubleshooting sign-in issues in the *AWS Sign-In User Guide*.

I lost my access keys

Access keys consist of two parts:

- **The access key identifier**. This is not a secret, and can be seen in the IAM console wherever access keys are listed, such as on the user summary page.
- **The secret access key**. This is provided when you initially create the access key pair. Just like a password, it ***cannot be retrieved later***. If you lost your secret access key, then you must create a new access key pair. If you already have the maximum number of access keys, you must delete an existing pair before you can create another.

If you lose your secret access key, you must delete the access key and create a new one. For more instructions, see Update access keys.

Policy variables aren't working

If your policy variables are not working, one of the following errors has occurred:

The date is wrong in the Version policy element.

Verify that all policies that include variables include the following version number in the policy: `"Version": "2012-10-17"`. Without the correct version number, the variables are not replaced during evaluation. Instead, the variables are evaluated literally. Policies that don't include variables still work when you include the latest version number.

A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version is created when you modify a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. To learn more about the `Version` policy element see IAM JSON policy elements: Version. To learn more about policy versions, see the section called “Versioning IAM policies”.

Variable characters are in the wrong letter case.

Verify that your policy variables are in the right case. For details, see IAM policy elements: Variables and tags.

Changes that I make are not always immediately visible

As a service that is accessed through computers in data centers around the world, IAM uses a distributed computing model called eventual consistency. Any changes that you make in IAM (or

other AWS services), including attribute-based access control (ABAC) tags, take time to become visible from all possible endpoints. Some delay results from the time it takes to send data from server to server, replication zone to replication zone, and Region to Region. IAM also uses caching to improve performance, but in some cases this can add time. The change might not be visible until the previously cached data times out.

You must design your global applications to account for these potential delays. Ensure that they work as expected, even when a change made in one location is not instantly visible at another. Such changes include creating or updating users, groups, roles, or policies. We recommend that you do not include such IAM changes in the critical, high availability code paths of your application. Instead, make IAM changes in a separate initialization or setup routine that you run less frequently. Also, be sure to verify that the changes have been propagated before production workflows depend on them.

For more information about how some other AWS services are affected by this, consult the following resources:

- **Amazon DynamoDB**: Read consistency in the *DynamoDB Developer Guide*, and Read Consistency in the Amazon DynamoDB Developer Guide.
- **Amazon EC2**: EC2 Eventual Consistency in the *Amazon EC2 API Reference*.
- **Amazon EMR**: Ensuring Consistency When Using Amazon S3 and Amazon EMR for ETL Workflows in the AWS Big Data Blog
- **Amazon Redshift**: Managing Data Consistency in the *Amazon Redshift Database Developer Guide*
- **Amazon S3**: Amazon S3 Data Consistency Model in the *Amazon Simple Storage Service User Guide*

I am not authorized to perform: iam:DeleteVirtualMFADevice

You might receive the following error when you attempt to assign or remove a virtual MFA device for yourself or others:

```
User: arn:aws:iam::123456789012:user/Diego is not authorized to
 perform: iam:DeleteVirtualMFADevice on resource: arn:aws:iam::123456789012:mfa/Diego
 with an explicit deny
```

This could happen if someone previously began assigning a virtual MFA device to a user in the IAM console and then cancelled the process. This creates a virtual MFA device for the user in IAM but

never assigns it to the user. Delete the existing virtual MFA device before you create a new virtual MFA device with the same device name.

To fix this issue, an administrator should **not** edit policy permissions. Instead, the administrator must use the AWS CLI or AWS API to delete the existing but unassigned virtual MFA device.

To delete an existing but unassigned virtual MFA device

1. View the virtual MFA devices in your account.
   - AWS CLI: `aws iam list-virtual-mfa-devices`
   - AWS API: `ListVirtualMFADevices`
2. In the response, locate the ARN of the virtual MFA device for the user you are trying to fix.
3. Delete the virtual MFA device.
   - AWS CLI: `aws iam delete-virtual-mfa-device`
   - AWS API: `DeleteVirtualMFADevice`

How do I securely create IAM users?

If you have employees that require access to AWS, you might choose to create IAM users or use IAM Identity Center for authentication. If you use IAM, AWS recommends that you create an IAM user and securely communicate the credentials to the employee. If you are not physically located next to your employee, use a secure workflow to communicate credentials to employees.

Use the following secure workflow to create a new user in IAM:

1. Create a new user using the AWS Management Console. Choose to grant AWS Management Console access with a generated password. If necessary, select the **Users must create a new password at next sign-in** check box. Do not add a permissions policy to the user until after they have changed their password.
2. After the user is added, copy the sign-in URL, user name, and password for the new user. To view the password, choose **Show**.
3. Send the password to your employee using a secure communications method in your company, such as email, chat, or a ticketing system. Separately, provide your users with the IAM user console link and their user name. Tell the employee to confirm that they can sign in successfully before you will grant them permissions.

4. After the employee confirms, add the permissions that they need. As a security best practice, add a policy that requires the user to authenticate using MFA to manage their credentials. For an example policy, see AWS: Allows MFA-authenticated IAM users to manage their own credentials on the Security credentials page.

Additional resources

The following resources can help you troubleshoot as you work with AWS.

- **AWS CloudTrail User Guide** – Use AWS CloudTrail to track a history of API calls made to AWS and store that information in log files. This helps you determine which users and accounts accessed resources in your account, when the calls were made, what actions were requested, and more. For more information, see Logging IAM and AWS STS API calls with AWS CloudTrail.
- **AWS Knowledge Center** – Find FAQs and links to other resources to help you troubleshoot issues.
- **AWS Support Center** – Get technical support.
- **AWS Premium Support Center** – Get premium technical support.

Troubleshoot access denied error messages

The following information can help you identify, diagnose, and resolve access denied errors with AWS Identity and Access Management. Access denied errors appear when AWS explicitly or implicitly denies an authorization request.

- An *explicit denial* occurs when a policy contains a `Deny` statement for the specific AWS action.
- An *implicit denial* occurs when there is no applicable `Deny` statement and also no applicable `Allow` statement. Because an IAM policy denies an IAM principal by default, the policy must explicitly allow the principal to perform an action. Otherwise, the policy implicitly denies access. For more information, see The difference between explicit and implicit denies.

When you make a request to a service or resource, multiple policies may apply to the request. Review all applicable policies in addition to the policy specified in the error message.

- If multiple policies of the same policy type deny a request, the access denied error message doesn't specify the number of policies evaluated.

- If multiple policy types deny an authorization request, AWS includes only one of those policy types in the error message.

> **Important**
>
> **Having trouble signing in to AWS?** Make sure that you're on the correct AWS sign-in page for your type of user. If you are the AWS account root user (account owner), you can sign in to AWS using the credentials that you set up when you created the AWS account. If you are an IAM user, your account administrator can give you AWS sign-in credentials. If you need to request support, do not use the feedback link on this page. The form is received by the AWS Documentation team, not Support. Instead, on the Contact Us page choose **Still unable to log into your AWS account** and then choose one of the available support options.

I get "access denied" when I make a request to an AWS service

- Check if the error message includes the type and Amazon Resource Name (ARN) of the policy responsible for denying access. If this is the case, then check for deny statements for the action in the specified policy. If the policy type is provided but there is no policy ARN, then focus on troubleshooting issues for that policy type: Check for deny statements for the action in policies of the specified type. If the error message doesn't mention the policy type responsible for denying access, use the rest of the guidelines in this section to troubleshoot further.
- Verify that you have the identity-based policy permission to call the action and resource that you have requested. If any conditions are set, you must also meet those conditions when you send the request. For information about viewing or modifying policies for an IAM user, group, or role, see Manage IAM policies.
- If the AWS Management Console returns a message stating that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator provided you with your sign-in credentials or sign-in link.

  The following example error occurs when the `mateojackson` IAM user attempts to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `widgets:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 widgets:GetWidget on resource: my-example-widget
```

In this case, Mateo must ask his administrator to update his policies to allow access to the *my-example-widget* resource using the `widgets:GetWidget` action.

- Are you trying to access a service that supports resource-based policies, such as Amazon S3, Amazon SNS, or Amazon SQS? If so, verify that the policy specifies you as a principal and grants you access. If you make a request to a service within your account, either your identity-based policies or the resource-based policies can grant you permission. If you make a request to a service in a different account, then both your identity-based policies and the resource-based policies must grant you permission. To view the services that support resource-based policies, see AWS services that work with IAM.
- If your policy includes a condition with a key–value pair, review it carefully. Examples include the `aws:RequestTag/tag-key` global condition key, the AWS KMS `kms:EncryptionContext:encryption_context_key`, and the `ResourceTag/tag-key` condition key supported by multiple services. Make sure that the key name does not match multiple results. Because condition key names are not case sensitive, a condition that checks for a key named `foo` matches `foo`, `Foo`, or `F00`. If your request includes multiple key–value pairs with key names that differ only by case, then your access might be unexpectedly denied. For more information, see IAM JSON policy elements: Condition.
- If you have a permissions boundary, verify that the policy that is used for the permissions boundary allows your request. If your identity-based policies allow the request, but your permissions boundary does not, then the request is denied. A permissions boundary controls the maximum permissions that an IAM principal (user or role) can have. Resource-based policies are not limited by permissions boundaries. Permissions boundaries are not common. For more information about how AWS evaluates policies, see Policy evaluation logic.
- If you are signing requests manually (without using the AWS SDKs), verify that you have correctly signed the request.
- If you're using an Amazon VPC endpoint policy and you get an access denied error that is not logged in AWS CloudTrail, it might be because the VPC endpoint owner account is different from the calling account or target role account.

I get "access denied" when I make a request with temporary security credentials

- First, make sure that you are not denied access for a reason that is unrelated to your temporary credentials. For more information, see I get "access denied" when I make a request to an AWS service.
- Verify that the service accepts temporary security credentials, see AWS services that work with IAM.
- Verify that your requests are being signed correctly and that the request is well-formed. For details, see your toolkit documentation or Use temporary credentials with AWS resources.
- Verify that your temporary security credentials haven't expired. For more information, see Temporary security credentials in IAM.
- Verify that the IAM user or role has the correct permissions. Permissions for temporary security credentials are derived from an IAM user or role. As a result, the permissions are limited to those that are granted to the role whose temporary credentials you have assumed. For more information about how permissions for temporary security credentials are determined, see Permissions for temporary security credentials.
- If you assumed a role, your role session might be limited by session policies. When you request temporary security credentials programmatically using AWS STS, you can optionally pass inline or managed session policies. Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary credential session for a role. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Alternatively, if your administrator or a custom program provides you with temporary credentials, they might have included a session policy to limit your access.
- If you are an AWS STS federated user principal, your session might be limited by session policies. You create a federated user session by signing in to AWS as an IAM user and then requesting a federation token. For more information, see Requesting credentials through a custom identity broker. If you or your identity broker passed session policies while requesting a federation token, then your session is limited by those policies. The resulting session's permissions are the intersection of your IAM user identity-based policies and the session policies. For more information about session policies, see Session policies.

- If you are accessing a resource that has a resource-based policy by using a role, verify that the policy grants permissions to the role. For example, the following policy allows `MyRole` from account `111122223333` to access `amzn-s3-demo-bucket`.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Sid": "S3BucketPolicy",
    "Effect": "Allow",
    "Principal": {"AWS": ["arn:aws:iam::111122223333:role/MyRole"]},
    "Action": ["s3:PutObject"],
    "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/*"]
  }]
}
```

Access denied error message examples

Most access denied error messages appear in the format `User` *`user`* `is not authorized to perform` *`action`* `on` *`resource`* `because` *`context`*. In this example, *`user`* is the ARN of the principal that is denied access, *`action`* is the service action that the policy denies, and *`resource`* is the ARN of the resource on which the policy acts. The *`context`* field provides additional context about the policy that denied access.

When a policy explicitly denies access because the policy contains a `Deny` statement, then AWS includes the phrase `with an explicit deny in a` *`type`* `policy` in the access denied error message. This phrase may also specify the ARN of the policy, as follows: `with an explicit deny in a` *`type`* `policy:` *`policy ARN`*. No ARN is present when an inline policy triggers the denial.

When the policy implicitly denies access, then AWS includes the phrase `because no` *`type`* `policy allows the` *`action`* `action` in the access denied error message.

> **Note**
>
> Some AWS services do not support this access denied error message format. The content of access denied error messages can vary depending on the service making the authorization request.

The following examples show the format for different types of access denied error messages.

Access denied due to a service control policy – implicit denial

1. Check for a missing `Allow` statement for the action in your service control policies (SCPs). For the following example, the action is `codecommit:ListRepositories`.
2. Update your SCP by adding the `Allow` statement. For more information, see Updating an SCP in the *AWS Organizations User Guide*.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codecommit:ListRepositories
because no service control policy allows the codecommit:ListRepositories action
```

Access denied due to a service control policy – explicit denial

1. If a policy ARN is provided in the error message, check for a `Deny` statement for the action in the specified service control policy (SCP). In the example below, the action is `codecommit:ListRepositories`.
2. If no policy ARN is provided in the error message, check for a `Deny` statement for the action in your SCPs.
3. Update your SCP by removing the `Deny` statement. For more information, see Update a service control policy (SCP) in the *AWS Organizations User Guide*.

Error message with a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codecommit:ListRepositories
with an explicit deny in a service control policy:
 arn:aws:organizations::777788889999:policy/o-exampleorgid/service_control_policy/p-
examplepolicyid123
```

Error message without a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codecommit:ListRepositories
with an explicit deny in a service control policy
```

Access denied due to a resource control policy – explicit denial

1. If a policy ARN is provided in the error message, check for a `Deny` statement for the action in the specified resource control policy (RCP). In the example below, the action is `secretsmanager:GetSecretValue`.
2. If no policy ARN is provided in the error message, check for a `Deny` statement for the action in your RCPs.
3. Update your RCP by removing the `Deny` statement. For more information, see Update a resource control policy (RCP) in the *AWS Organizations User Guide*.

Error message with a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 secretsmanager:GetSecretValue
on resource: arn:aws:secretsmanager:us-east-1:123456789012:secret:*
with an explicit deny in a resource control policy:
 arn:aws:organizations::777788889999:policy/o-exampleorgid/resource_control_policy/p-
examplepolicyid456
```

Error message without a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 secretsmanager:GetSecretValue
on resource: arn:aws:secretsmanager:us-east-1:123456789012:secret:*
with an explicit deny in a resource control policy
```

Access denied due to a VPC endpoint policy – implicit denial

1. Check for a missing `Allow` statement for the action in your Virtual Private Cloud (VPC) endpoint policies. For the following example, the action is `codecommit:ListRepositories`.
2. Update your VPC endpoint policy by adding the `Allow` statement. For more information, see Update a VPC endpoint policy in the *AWS PrivateLink Guide*.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codecommit:ListRepositories
because no VPC endpoint policy allows the codecommit:ListRepositories action
```

Access denied due to a VPC endpoint policy – explicit denial

1. Check for an explicit `Deny` statement for the action in your Virtual Private Cloud (VPC) endpoint policies. For the following example, the action is `codedeploy:ListDeployments`.
2. Update your VPC endpoint policy by removing the `Deny` statement. For more information, see Update a VPC endpoint policy in the *AWS PrivateLink Guide*.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codedeploy:ListDeployments
on resource: arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:*
with an explicit deny in a VPC endpoint policy
```

Access denied due to a permissions boundary – implicit denial

1. Check for a missing `Allow` statement for the action in your permissions boundary. For the following example, the action is `codedeploy:ListDeployments`.
2. Update your permissions boundary by adding the `Allow` statement to your IAM policy. For more information, see Permissions boundaries for IAM entities and Edit IAM policies.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codedeploy:ListDeployments
on resource: arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:*
because no permissions boundary allows the codedeploy:ListDeployments action
```

Access denied due to a permissions boundary – explicit denial

1. If a policy ARN is provided in the error message, check for a `Deny` statement for the action in the specified permissions boundary. In the example below, the action is `sagemaker:ListModels`.
2. If no policy ARN is provided in the error message, check for a `Deny` statement for the action in the permissions boundary attached to the principal.
3. Update your permissions boundary by removing the `Deny` statement from your IAM policy. For more information, see Permissions boundaries for IAM entities and Edit IAM policies.

Error message with a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 sagemaker:ListModels
with an explicit deny in a permissions boundary: arn:aws:iam::123456789012:policy/
DeveloperPermissionBoundary
```

Error message without a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 sagemaker:ListModels
with an explicit deny in a permissions boundary
```

Access denied due to session policies – implicit denial

1. Check for a missing `Allow` statement for the action in your session policies. For the following example, the action is `codecommit:ListRepositories`.
2. Update your session policy by adding the `Allow` statement. For more information, see Session policies and Edit IAM policies.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codecommit:ListRepositories
because no session policy allows the codecommit:ListRepositories action
```

Access denied due to session policies – explicit denial

1. If a policy ARN is provided in the error message, check for a `Deny` statement for the action in the specified session policy. In the example below, the action is `codedeploy:ListDeployments`.
2. If no policy ARN is provided in the error message, check for a `Deny` statement for the action in your session policies.
3. Update your session policy by removing the `Deny` statement. For more information, see Session policies and Edit IAM policies.

Error message with a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codedeploy:ListDeployments
on resource: arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:*
```

```
with an explicit deny in a session policy: arn:aws:iam::123456789012:policy/
DeveloperSessionPolicy
```

Error message without a policy ARN:

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 codedeploy:ListDeployments
on resource: arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:*
with an explicit deny in a session policy
```

Access denied due to resource-based policies – implicit denial

1. Check for a missing `Allow` statement for the action in your resource-based policy. For the following example, the action is `secretsmanager:GetSecretValue`.
2. Update your policy by adding the `Allow` statement. For more information, see Resource-based policies and Edit IAM policies.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 secretsmanager:GetSecretValue
because no resource-based policy allows the secretsmanager:GetSecretValue action
```

Access denied due to resource-based policies – explicit denial

1. Check for an explicit `Deny` statement for the action in your resource-based policy. For the following example, the action is `secretsmanager:GetSecretValue`.
2. Update your policy by removing the `Deny` statement. For more information, see Resource-based policies and Edit IAM policies.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform:
 secretsmanager:GetSecretValue
on resource: arn:aws:secretsmanager:us-east-1:123456789012:secret:*
with an explicit deny in a resource-based policy
```

Access denied due to role trust policies – implicit denial

1. Check for a missing `Allow` statement for the action in your role trust policy. For the following example, the action is `sts:AssumeRole`.

2. Update your policy by adding the `Allow` statement. For more information, see Resource-based policies and Edit IAM policies.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform: sts:AssumeRole
because no role trust policy allows the sts:AssumeRole action
```

Access denied due to role trust policies – explicit denial

1. Check for an explicit `Deny` statement for the action in your role trust policy. For the following example, the action is `sts:AssumeRole`.
2. Update your policy by removing the `Deny` statement. For more information, see Resource-based policies and Edit IAM policies.

```
User: arn:aws:iam::123456789012:user/John is not authorized to perform: sts:AssumeRole
with an explicit deny in the role trust policy
```

Access denied due to identity-based policies – implicit denial

1. Check for a missing `Allow` statement for the action in identity-based policies attached to the identity. For the following example, the action is `codecommit:ListRepositories` attached to the role HR.
2. Update your policy by adding the `Allow` statement. For more information, see Identity-based policies and Edit IAM policies.

```
User: arn:aws:iam::123456789012:role/HR is not authorized to perform:
 codecommit:ListRepositories
because no identity-based policy allows the codecommit:ListRepositories action
```

Access denied due to identity-based policies – explicit denial

1. If a policy ARN is provided in the error message, check for a `Deny` statement for the action in the specified policy. In the example below, the action is `codedeploy:ListDeployments`.
2. If no policy ARN is provided in the error message, check for a `Deny` statement for the action in identity-based policies attached to the identity.

3. Update your policy by removing the Deny statement. For more information, see Identity-based policies and Edit IAM policies.

Error message with a policy ARN:

```
User: arn:aws:iam::123456789012:role/HR is not authorized to perform:
 codedeploy:ListDeployments
on resource: arn:aws:codedeploy:us-east-1:123456789012:deploymentgroup:*
with an explicit deny in an identity-based policy: arn:aws:iam::123456789012:policy/
HRAccessPolicy
```

Error message without a policy ARN:

```
User: arn:aws:iam::123456789012:role/HR is not authorized to perform:
 codedeploy:ListDeployments
on resource: arn:aws:codedeploy:us-east-1:123456789012:deployment group:*
with an explicit deny in an identity-based policy
```

Troubleshoot access denied error messages with authorization ID (Preview)

Important

Access Troubleshooter is in public preview. Its APIs are subject to change.

Overview

When AWS denies a request for a supported API, the access denied error response usually includes an authorization ID. This ID references the authorization evaluation for that denied request. To retrieve the details of the evaluation and identify what caused the denial, make a request to the `GetRequestAuthorizationDetails` API and provide the authorization ID, or open the link present in the error message. This lets you identify the cause of a denied request in a single call, rather than resolving denials one at a time.

> **Note**
>
> AWS does not guarantee an authorization ID for each denied request. See When you don't receive an authorization ID in this document.

The following example shows an access denied error message that includes a policy ARN, a link to the AWS Management Console, and an authorization ID:

```
User: arn:aws:iam::123456789012:user/janedoe is not authorized to perform:
 iam:CreateInstanceProfile
on resource: arn:aws:iam::123456789012:instance-profile/my-instance-profile
with an explicit deny in an identity-based policy: arn:aws:iam::123456789012:policy/
ExampleDenyPolicy.
Go to https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/
authorization-details/EXAMPLE1a2b3c4d5e6f7g8h9i0j for complete details,
or call the GetRequestAuthorizationDetails API with the following authorization id:
 EXAMPLE1a2b3c4d5e6f7g8h9i0j
```

Concepts and definitions

- **Authorization details.** The authorization details include the request context, the evaluations performed, and the policies that were evaluated.
- **Authorization ID.** A unique identifier included in a supported access denied error response. It references the authorization evaluation for that denied request, and is the value you pass to the AWS Management Console link or the `GetRequestAuthorizationDetails` API.
- **Request context.** The values of the condition context keys evaluated for the request. This includes global condition context keys and service-specific condition context keys.
- **Evaluation.** For each action/resource pair in the request, the condition context keys that applied and the evaluated result: `ALLOW`, `EXPLICIT_DENY` (a policy explicitly denied the action), or `IMPLICIT_DENY` (no policy allowed the action), and a reference to each policy and statement that matched the action/resource pair. Each reference is a policy ARN (for managed policies) or a URI (for inline and other non-managed policies) that is reused throughout the response.
- **Policies evaluated during authorization.** The policies AWS considered for the request, including policies that did not match the action/resource pair. Each entry gives the policy type (for

example, identity-based, resource-based, SCP, RCP, permissions boundary, or session policy) and the same ARN (for managed policies) or URI used in the evaluations (for inline and other non-managed policies). Use the ARN or URI to match a policy back to the evaluation that referenced it. The “Attached to” field shows the principal ARN of the principal that the evaluated policy is attached to.

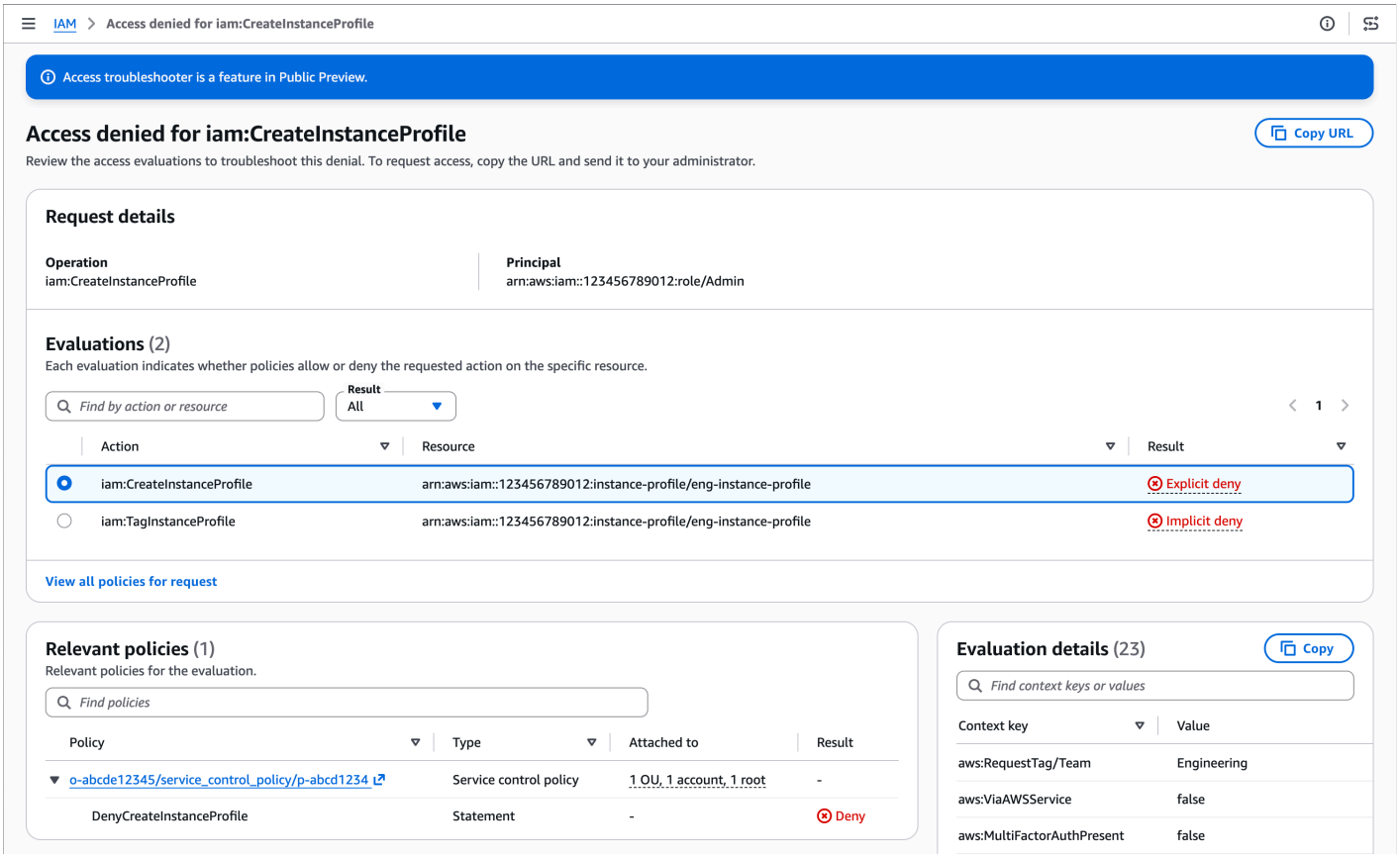
Use these results together to identify which policy denied access and why. The evaluation reflects the policies as they were at the time of the denial. Policies might have changed since then.

AWS automatically includes an authorization ID in access denied error responses for supported APIs, but authorization IDs are not guaranteed. In some cases, AWS does not record the authorization details for a denial, and the error response does not include an authorization ID. When details are recorded, they might not be available immediately, because AWS records them asynchronously. If a call to `GetRequestAuthorizationDetails` returns a not-found response, wait a short time and try again. AWS retains authorization details for 24 hours after the denial.

Troubleshoot using the AWS Management Console

Choose the link in the access denied error message to open the authorization details in the Access Troubleshooter Console.

- **What to look for.** The Access Troubleshooter Console shows the evaluation for each action/resource pair in the request, the result of each one (`Allow`, `Explicit deny`, or `Implicit deny`), and the policies that were considered. Find the pair that was denied and the policy that caused the denial. See the following screenshoot as an example:



- **Determine whether the denial was intended.** Review the evaluation before changing a policy. A denial can be correct, and the cause is sometimes the request context rather than the policy. For example, a request might be missing an expected session tag or coming from an unexpected network path.
- **How to resolve the denial.** Locate the corresponding policy and update it to resolve the access issue. For an explicit deny, remove or narrow the `Deny` statement for the action. For an implicit deny, add an `Allow` statement for the action. For more information, see Edit IAM policies.

Who can access the authorization details. For a request within a single account or a single organization, anyone who is granted access to `iam:GetRequestAuthorizationDetails` can access the details of the request. For a request that spans more than one organization, a user in a given organization can view only the details that pertain to their own organization.

Troubleshoot using AWS API or AWS CLI

To troubleshoot a denial using the AWS API or AWS CLI:

1. Locate the authorization ID in the access denied error response.

2. Call `GetRequestAuthorizationDetails` with the authorization ID, in the same AWS Region where the denial occurred. The following example shows the AWS CLI command:

```
aws iam-toolbox get-request-authorization-details \
    --region us-east-1 \
    --authorization-id EXAMPLE1a2b3c4d5e6f7g8h9i0j
```

3. Review the evaluation results to identify which policies denied access and the reason for the denial.
4. Update the identified policies to resolve the access issue. For more information, see Edit IAM policies.

When you don't receive an authorization ID

AWS does not return an authorization ID in the following cases:

- **The authorization is too large to record.** When a request involves a large number of policies, AWS may decide to not record the request.
- **The service isn't supported yet.** See Supported services.
- **The API isn't supported.** Some APIs within a supported service don't return an authorization ID.
- **The request type isn't supported.** This includes requests made by a service on your behalf and anonymous requests.

If you don't receive an authorization ID, see Troubleshoot access denied error messages for other ways to diagnose the denial.

Additional considerations

Required permissions. To call `GetRequestAuthorizationDetails`, you must have permissions for the `iam:GetRequestAuthorizationDetails` action.

Regional API. `GetRequestAuthorizationDetails` is a regional API. Call it in the same AWS Region where the access denied error and authorization ID were generated.

Cross-organization requests. When a denied request involves more than one organization, `GetRequestAuthorizationDetails` returns only the details that belong to your own

organization. This applies in both directions: If you are in the organization of the denied caller, you see only the details that pertain to that organization. If you are in the organization of the denied resource, you see only the details that pertain to that organization. Details owned by another organization, such as that organization's policies or context key values, are not returned.

If you get an access denied error from an API, service or region that does not yet include an authorization ID, see Troubleshoot access denied error messages.

Supported services

Support for Access Troubleshooter is rolling out gradually across AWS services and AWS Region. Some of the APIs for the following services support Access Troubleshooter:

- AWS CodeDeploy
- AWS Identity and Access Management
- AWS Key Management Service
- AWS Security Token Service
- AWS Step Functions
- Amazon Elastic Compute Cloud
- Amazon Elastic Container Registry
- Amazon Kinesis Data Streams
- Amazon Simple Queue Service

Troubleshoot issues with the root user

Use the information here to help you troubleshoot issues related to the root user of an AWS account.

> **Note**
>
> AWS accounts managed using AWS Organizations may have centralized root access enabled for member accounts. These member accounts do not have root user credentials, can't sign in as a root user, and are prevented from recovering the root user password. Contact your administrator if you need to perform a task that requires root user credentials.

I can't perform tasks that I expect to be able to do when signed in as the account root user

Your account might be a member of an organization in AWS Organizations. Your organizational administrator may have a service control policy (SCP) to limit the permissions of your account. SCPs impact all users, including the root user. For more information, see Service control policies in the *AWS Organizations User Guide*.

I forgot the root user password for my AWS account

If you're a root user and you have lost or forgot the password for your AWS account, you can reset your password. You must know the email address used to create the AWS account, and you must have access to the email account. For more information, see Reset a lost or forgotten root user password.

I don't have access to the email for my AWS account

When you create an AWS account, you provide an email address and password. These are the credentials for the AWS account root user. If you aren't sure of the email address associated with your AWS account, search for messages sent from `@signin.aws` or `@verify.signin.aws` to any email address for your organization that might have been used to open the AWS account.

If you know the email address but no longer have access to the email, try to recover access to the email. Use one of the following options to regain access to your email:

- If you own the domain for the email address, you can restore a deleted email address. Alternatively, you can set up a catch-all for your email account. A catch-all collects all messages sent to email addresses that no longer exist in the mail server and redirects them to another email address.
- If the email address on the account is part of your corporate email system, we recommend that you contact your IT system administrators. They might be able to help you regain access to the email.

If you're still not able to sign in to your AWS account, you can find alternate support options at Contact us.

Troubleshoot IAM policies

A policy is an entity in AWS that, when attached to an identity or resource, defines their permissions. AWS evaluates these policies when a principal, such as a user, makes a request. Permissions in the policies determine whether the request is allowed or denied.

Policies are stored in AWS as JSON documents that are attached to principals as *identity-based policies* or to resources as *resource-based policies*. Identity-based policies include AWS managed policies, customer managed policies, and inline policies.

You can create and edit customer managed policies in the AWS Management Console using both **Visual** and **JSON** editor options. When you view a policy in the AWS Management Console, you can see a summary of the permissions that are granted by that policy. You can use the visual editor and policy summaries to help you diagnose and fix common errors encountered while managing IAM policies.

Keep in mind that all IAM policies are stored using syntax that begins with the rules of JavaScript Object Notation (JSON). You do not have to understand this syntax to create or manage your policies. You can create and edit a policy using the visual editor in the AWS Management Console. To learn more about JSON syntax in IAM policies, see Grammar of the IAM JSON policy language.

Troubleshooting IAM Policy Topics

- Troubleshoot using the visual editor
  - Policy restructuring
  - Choosing a resource ARN in the visual editor
  - Denying permissions in the visual editor
  - Specifying multiple services in the visual editor
  - Reducing the size of your policy in the visual editor
  - Fixing unrecognized services, actions, or resource types in the visual editor
- Troubleshoot with policy summaries
  - Missing policy summary
  - Policy summary includes unrecognized services, actions, or resource types
  - Service does not support IAM policy summaries
  - My policy does not grant the expected permissions
- Troubleshoot policy management

- Attaching or detaching a policy in an IAM account
- Changing policies for your IAM identities based on their activity
- Troubleshoot JSON policy documents
  - Validate your policies
  - I don't have permissions for policy validation in the JSON editor
  - More than one JSON policy object
  - More than one JSON statement element
  - More than one effect, action, or resource element in a JSON statement element
  - Missing JSON version element

Troubleshoot using the visual editor

When you create or edit a customer managed policy, you can use information in the **Visual** editor to help you troubleshoot errors in your policy. To view an example of using the visual editor to create a policy, see the section called “Controlling access to identities”.

Policy restructuring

When you create a policy, AWS validates, processes, and transforms the policy before storing it. When the policy is retrieved, AWS transforms it back to a human-readable format without changing permissions. This can result in differences in what you see in the policy visual editor or **JSON** tab.

- Visual editor permission blocks can be added, removed, or reordered, and content within a block can be optimized.
- In the **JSON** tab, insignificant white space can be removed, and elements within JSON maps can be reordered. In addition, AWS account IDs within the principal elements can be replaced by the Amazon Resource Name (ARN) of the AWS account root user.

Because of these possible changes, you should not compare JSON policy documents as strings.

When you create a customer managed policy in the AWS Management Console, you can choose to work entirely in the **JSON** editor. If you never change the policy in the **Visual** editor and choose **Next** from the **JSON** editor, the policy is less likely to be restructured. When you use the **Visual** editor, IAM might restructure the policy to optimize its appearance. This restructuring exists only in your editing session and is not saved automatically.

If your policy is restructured in your editing session, IAM determines whether to save the restructuring based on the following situations:

Using this editor option	If you edit your policy	And then choose *Next* from this tab	When you choose *Save changes*
Visual	Edited	**Visual**	The policy is restructured
Visual	Edited	**JSON**	The policy is restructured
Visual	Not Edited	**Visual**	The policy is restructured
JSON	Edited	**Visual**	The policy is restructured
JSON	Edited	**JSON**	The policy structure is not changed
JSON	Not Edited	**JSON**	The policy structure is not changed

IAM might restructure complex policies or policies that have permission blocks or statements that allow multiple services, resource types, or condition keys.

Choosing a resource ARN in the visual editor

When you create or edit a policy using the visual editor, you must first choose a service, and then choose actions from that service. If the service and actions that you selected support choosing specific resources, then the visual editor lists the supported resource types. You can then choose **Add ARN** to provide the details about your resource. You can choose from the following options for adding an ARN for a resource type.

- **Use the ARN builder** – You might see different fields to build your ARN based on the resource type. You can also choose **Any** to provide permissions for any value for the specified setting. For example, if you selected the Amazon EC2 **Read** access level group, then the actions in your policy support the `instance` resource type. Provide the **Region**, **Account**, and **InstanceId** values for

your resource. The policy grants permissions to any instance in your account if you provide your account ID and choose **Any** for the Region and instance ID.

- **Type or paste the ARN** – You can specify resources by their Amazon Resource Name (ARN). You can include a wildcard character (*) in any field of the ARN (between each pair of colons). For more information, see IAM JSON policy elements: Resource.

Denying permissions in the visual editor

By default, the policy that you create using the visual editor allows the actions that you choose. To deny the chosen actions instead, choose **Switch to deny permissions**. Because requests are *denied by default*, we recommend that you allow permissions to only those actions and resources that a user needs. You should create a deny statement only if you want to override a permission separately that is allowed by another statement or policy. We recommend that you limit the number of deny permissions to a minimum because they can increase the difficulty of troubleshooting permissions. For more information about how IAM evaluates policy logic, see Policy evaluation logic.

> **Note**
>
> By default, only the AWS account root user has access to all the resources in that account. So if you are not signed in as the root user, you must have permissions granted by a policy.

Specifying multiple services in the visual editor

When you use the visual editor to construct a policy, you can select only one service at a time. This is a best practice because the visual editor then allows you to choose from the actions for that one service. You then choose from the resources supported by that service and the selected actions. This makes it easier to create and troubleshoot your policy.

You can also use a wildcard character (*) to manually specify multiple services. For example, type **`Code*`** to provide permissions for all services beginning with `Code`, such as `CodeBuild` and `CodeCommit`. However, you must then type the actions and resource ARNs to complete your policy. Additionally, when you save your policy, it might be restructured to include each service in a separate permission block.

Alternatively, to use JSON syntax (such as wildcards) for services, create, edit, and save your policy using the **JSON** editor option.

Reducing the size of your policy in the visual editor

When you use the visual editor to create a policy, IAM creates a JSON document to store your policy. You can view this document by switching to the **JSON** editor option. If this JSON document exceeds the size limit of a policy, the visual editor displays an error message. You will not be able to review and save the policy. To view the IAM limitation on the size of a managed policy, see IAM and STS character limits.

To reduce the size of your policy in the visual editor, edit your policy or move permission blocks to another policy. The error message includes the number of characters that your policy document contains. You can use this information to help you reduce the size of your policy.

Fixing unrecognized services, actions, or resource types in the visual editor

You might see a warning in the visual editor that your policy includes an unrecognized service, action, or resource type.

> **Note**
>
> IAM reviews service names, actions, and resource types for services that support policy summaries. However, your policy summary might include a resource value or condition that does not exist. Always test your policies with the policy simulator.

If your policy includes unrecognized services, actions or resource types, one of the following errors has occurred:

- **Preview service** – Services that are in preview do not support the visual editor. If you are participating in the preview, you must manually type the actions and resource ARNs to complete your policy. You can ignore any warnings and continue. Alternatively, you can choose the **JSON** editor option to type or paste a JSON policy document.
- **Custom service** – Custom services do not support the visual editor. If you are using a custom service, you must manually type the actions and resource ARNs to complete your policy. You can ignore any warnings and continue. Alternatively, you can choose the **JSON** editor option to type or paste a JSON policy document.
- **Service does not support the visual editor** – If your policy includes a generally available (GA) service that does not support the visual editor, you must manually type the actions and resource ARNs to complete your policy. You can ignore any warnings and continue. Alternatively, you can choose the **JSON** editor option to type or paste a JSON policy document.

Generally available services are services that are released publicly and are not preview or custom services. If an unrecognized service is generally available and the name is spelled correctly, then the service does not support the visual editor. To learn how to request visual editor or policy summary support for a GA service, see Service does not support IAM policy summaries.

- **Action does not support the visual editor** – If your policy includes a supported service with an unsupported action, you must manually type the actions and resource ARNs to complete your policy. You can ignore any warnings and continue. Alternatively, you can choose the **JSON** editor option to type or paste a JSON policy document.

  If your policy includes a supported service with an unsupported action, then the service does not fully support the visual editor. To learn how to request visual editor or policy summary support for a GA service, see Service does not support IAM policy summaries.
- **Resource type does not support the visual editor** – If your policy includes a supported action with an unsupported resource type, you can ignore the warning and continue. However, IAM cannot confirm that you have included resources for all of your selected actions, and you might see additional warnings.
- **Typo** – When you manually type a service, action, or resource in the visual editor, you can create a policy that includes a typo. We recommend that you use the visual editor by selecting from the list of services and actions. Then, complete the resource section according to the prompts. If a service does not fully support the visual editor, you might have to manually type parts of your policy.

  If you are certain that your policy contains none of the errors above, then your policy might include a typo. Check for the following issues:

  - Misspelled service, action, and resource type names, such as `s2` instead of `s3` or `ListMyBuckets` instead of `ListAllMyBuckets`
  - Unnecessary text in ARNs, such as `arn:aws:s3: : :*`
  - Missing colons in actions, such as `iam.CreateUser`

  You can evaluate a policy that might include typos by choosing **Next** to review the policy summary. Then, confirm whether the policy provides the permissions you intended.

Troubleshoot with policy summaries

You can diagnose and resolve issues related to policy summaries.

Missing policy summary

The IAM console includes *policy summary* tables that describe the access level, resources, and conditions that are allowed or denied for each service in a policy. Policies are summarized in three tables: the policy summary, the service summary, and the action summary.

The *policy summary* table includes a list of services and summaries of the permissions that are defined by the chosen policy. The policy summary is available on the **Policy details** page for any attached policy, or on the **Policies** page for managed policies. If AWS is unable to render a summary for a policy, the console displays the JSON policy document and the following error:

A summary for this policy cannot be generated. You can still view or edit the JSON policy document.

If your policy does not include a summary, one of the following errors has occurred:

- **Unsupported policy element** – IAM does not support generating policy summaries for policies that include one of the following policy elements:
  - `Principal`
  - `NotPrincipal`
  - `NotResource`
- **No policy permissions** – If a policy does not provide any effective permissions, then the policy summary cannot be generated. For example, if a policy includes a single statement with the element `"NotAction": "*"`, then it grants access to all actions except "all actions" (*). This means it grants `Deny` or `Allow` access to nothing.

> **Note**
>
> Be careful when using these policy elements such as `NotPrincipal`, `NotAction`, and `NotResource`. For information about using policy elements, see IAM JSON policy element reference.

If you provide mismatched services and resources, you can create a policy that does not provide effective permissions. This can occur when you specify actions in one service and resources from another service. In this case, the policy summary does appear. The only indication that there is a problem is that the resource column in the summary can include a resource from a different

service. If this column includes a mismatched resource, then you should review your policy for errors. Test your policies with the policy simulator to better understand the policy.

Policy summary includes unrecognized services, actions, or resource types

In the IAM console, if a policy summary includes a warning symbol (), then the policy might include an unrecognized service, action, or resource type. To learn about warnings within a policy summary, see Policy summary (list of services).

> **Note**
>
> IAM reviews service names, actions, and resource types for services that support policy summaries. However, your policy summary might include a resource value or condition that does not exist. Always test your policies with the policy simulator.

If your policy includes unrecognized services, actions or resource types, one of the following errors has occurred:

- **Preview service** – Services that are in preview do not support policy summaries.
- **Custom service** – Custom services do not support policy summaries.
- **Service does not support summaries** – If your policy includes a generally available (GA) service that does not support policy summaries, then the service is included in the **Unrecognized services** section of the policy summary table. Generally available services are services that are released publicly and are not preview or custom services. If an unrecognized service is generally available and the name is spelled correctly, then the service does not support IAM policy summaries. To learn how to request policy summary support for a GA service, see Service does not support IAM policy summaries.
- **Action does not support summaries** – If your policy includes a supported service with an unsupported action, then the action is included in the **Unrecognized actions** section of the service summary table. To learn about warnings within a service summary, see Service summary (list of actions).
- **Resource type does not support summaries** – If your policy includes a supported action with an unsupported resource type, then the resource is included in the **Unrecognized resource types**

section of the service summary table. To learn about warnings within a service summary, see Service summary (list of actions).

- **Typo** – AWS checks that the JSON is syntactically correct, and that the policy does not include typos or other errors as part of policy validation.

> **Note**
>
> As a best practice, we recommend that you use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions. We recommend that you open your existing policies and review and resolve any policy validation recommendations.

Service does not support IAM policy summaries

It is possible for the IAM policy summaries or the visual editor to not support a generally available (GA) service or action. Generally available services are services that are released publicly and are not previewed or custom services. If an unrecognized service is generally available and the name is spelled correctly, then the service does not support these features. If your policy includes a supported service with an unsupported action, then the service does not fully support IAM policy summaries.

To request that a service add IAM policy summary or visual editor support

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Locate the policy that includes the unsupported service:
   - If the policy is a managed policy, choose **Policies** in the navigation pane. In the list of policies, choose the name of the policy that you want to view.
   - If the policy is an inline policy attached to the user, choose **Users** in the navigation pane. In the list of users, choose the name of the user whose policy you want to view. In the table of policies for the user, expand the header for the policy summary that you want to view.
3. In the left side on the AWS Management Console footer, choose **Feedback**. In the **Feedback for IAM** box, type **`I request that the <ServiceName> service add support for IAM policy summaries and the visual editor`**. If you want more than one service to support summaries, type **`I request that the <ServiceName1>, <ServiceName2>,`**

and <ServiceName3> services add support for IAM policy summaries and the visual editor.

To request that a service add IAM policy summary support for a missing action

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Locate the policy that includes the unsupported service:
   - If the policy is a managed policy, choose **Policies** in the navigation pane. In the list of policies, choose the name of the policy that you want to view.
   - If the policy is an inline policy attached to the user, choose **Users** in the navigation pane. In the list of users, choose the name of the user whose policy you want to view. In the table of policies for the user, choose the name of the policy that you want to view to expand the policy summary.
3. In the policy summary, choose the name of the service that includes an unsupported action.
4. In the left side on the AWS Management Console footer, choose **Feedback**. In the **Feedback for IAM** box, type **I request that the <ServiceName> service add IAM policy summary and the visual editor support for the <ActionName> action**.
   If you want to report more than one unsupported action, type **I request that the <ServiceName> service add IAM policy summary and the visual editor support for the <ActionName1>, <ActionName2>, and <ActionName3> actions**.

To request that a different service includes missing actions, repeat the last three steps.

My policy does not grant the expected permissions

To assign permissions to a user, group, role, or resource, you create a *policy*, which is a document that defines permissions. The policy document includes the following elements:

- **Effect** – whether the policy allows or denies access
- **Action** – the list of actions that are allowed or denied by the policy
- **Resource** – the list of resources on which the actions can occur
- **Condition** (Optional) – the circumstances under which the policy grants permission

To learn about these and other policy elements, see IAM JSON policy element reference.

To grant access, your policy must define an action with a supported resource. If your policy also includes a condition, that condition must include a global condition key or must apply to the action. To learn which resources are supported by an action, see the AWS documentation for your service. To learn which conditions are supported by an action, see Actions, Resources, and Condition Keys for AWS Services.

Check whether your policy defines an action, resource, or condition that does not grant permissions. View the policy summary for your policy using the IAM console at https://console.aws.amazon.com/iam/. You can use policy summaries to identify and correct problems in your policy.

There are several reasons why an element might not grant permissions despite being defined in the IAM policy:

- **An action is defined without an applicable resource**
- **A resource is defined without an applicable action**
- **A condition is defined without an applicable action**

To view examples of policy summaries that include warnings, see the section called "Policy summary (list of services)".

An action is defined without an applicable resource

The policy below defines all `ec2:Describe*` actions and defines a specific resource. None of the `ec2:Describe` actions are granted because none of these actions support resource-level permissions. Resource-level permissions mean that the action supports resources using ARNs in the policy's `Resource` element. If an action does not support resource-level permissions, then that statement in the policy must use a wildcard (`*`) in the `Resource` element. To learn which services support resource-level permissions, see AWS services that work with IAM.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:Describe*",
            "Resource": "arn:aws:ec2:us-east-2:111122223333:instance/*"
```

```
        }
    ]
}
```

This policy does not provide any permissions, and the policy summary includes the following error:

`This policy does not grant any permissions. To grant access, policies must have an action that has an applicable resource or condition.`

To fix this policy, you must use * in the `Resource` element.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "ec2:Describe*",
        "Resource": "*"
    }]
}
```

A resource is defined without an applicable action

The policy below defines an Amazon S3 bucket resource but does not include an S3 action that can be performed on that resource. This policy also grants full access to all Amazon CloudFront actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Action": "cloudfront:*",
        "Resource": [
            "arn:aws:cloudfront:*",
            "arn:aws:s3:::amzn-s3-demo-bucket"
        ]
```

```
        }
    ]
}
```

This policy provides permissions for all CloudFront actions. But because the policy defines the S3 `amzn-s3-demo-bucket` resource without defining any S3 actions, the policy summary includes the following warning:

`This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition.`

To fix this policy to provide S3 bucket permissions, you must define S3 actions that can be performed on a bucket resource.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudfront:*",
                "s3:CreateBucket",
                "s3:ListBucket*",
                "s3:PutBucket*",
                "s3:GetBucket*"
            ],
            "Resource": [
                "arn:aws:cloudfront:*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

Alternately, to fix this policy to provide only CloudFront permissions, remove the S3 resource.

A condition is defined without an applicable action

The policy below defines two Amazon S3 actions for all S3 resources, if the S3 prefix equals `custom` and the version ID equals `1234`. However, the `s3:VersionId` condition key is used for object version tagging and is not supported by the defined bucket actions. To learn which conditions are supported by an action, see Actions, Resources, and Condition Keys for AWS Services and choose the service to view service documentation for condition keys.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucketVersions",
                "s3:ListBucket"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:prefix": [
                        "custom"
                    ],
                    "s3:VersionId": [
                        "1234"
                    ]
                }
            }
        }
    ]
}
```

This policy provides permissions for the `s3:ListBucketVersions` action and the `s3:ListBucket` action if the bucket name includes the `custom` prefix. But because the `s3:VersionId` condition is not supported by any of the defined actions, the policy summary includes the following error:

`This policy does not grant any permissions. To grant access, policies must have an action that has an applicable resource or condition.`

To fix this policy to use S3 object version tagging, you must define an S3 action that supports the `s3:VersionId` condition key.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucketVersions",
                "s3:ListBucket",
                "s3:GetObjectVersion"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:prefix": [
                        "custom"
                    ],
                    "s3:VersionId": [
                        "1234"
                    ]
                }
            }
        }
    ]
}
```

This policy provides permissions for every action and condition in the policy. However, the policy still does not provide any permissions because there is no case where a single action matches both conditions. Instead, you must create two separate statements that each include only actions with the conditions to which they apply.

To fix this policy, create two statements. The first statement includes the actions that support the `s3:prefix` condition, and the second statement includes the actions that support the `s3:VersionId` condition.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucketVersions",
                "s3:ListBucket"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:prefix": "custom"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "s3:GetObjectVersion",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:VersionId": "1234"
                }
            }
        }
    ]
}
```

Troubleshoot policy management

You can diagnose and resolve issues relating to policy management.

Attaching or detaching a policy in an IAM account

Some AWS managed policies are linked to a service. These policies are used only with a service-linked role for that service. In the IAM console, when you view the **Policy details** page, the page includes a banner to indicate that the policy is linked to a service. You cannot attach this policy to a user, group, or role within IAM. When you create a service-linked role for the service, this policy is automatically attached to your new role. Because the policy is required, you cannot detach the policy from the service-linked role.

Changing policies for your IAM identities based on their activity

You can update policies for your IAM identities (users, groups, and roles) based on their activity. To do this, view your account's events in CloudTrail **Event history**. CloudTrail event logs include detailed event information that you can use to change the policy's permissions.

A user or role is attempting to perform an action in AWS and that request is denied.

Consider whether the user or role should have permission to perform the action. If so, you can add the action and even the ARN of the resource that they attempted to access to their policy.

A user or role has permissions that they are not using.

Consider removing those permissions from their policy. Make sure that your policies grant the least privilege that is needed to perform only the necessary actions.

For more information about using CloudTrail, see Viewing CloudTrail Events in the CloudTrail Console in the *AWS CloudTrail User Guide*.

Troubleshoot JSON policy documents

You can diagnose and resolve issues relating to JSON policy documents.

Validate your policies

When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see IAM policy validation. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation.

I don't have permissions for policy validation in the JSON editor

In the AWS Management Console, you might receive the following error if you do not have permissions to view IAM Access Analyzer policy validation results:

`You need permissions. You do not have the permissions required to perform this operation. Ask your administrator to add permissions.`

To fix this error, ask your administrator to add the `access-analyzer:ValidatePolicy` permission for you.

More than one JSON policy object

An IAM policy must consist of only one JSON object. You denote an object by placing { } braces around it. You can nest other objects within a JSON object by embedding additional { } braces within the outer pair. A policy must contain only one outermost pair of { } braces. The following example is incorrect because it contains two objects at the top level (called out in *red*):

JSON

```
{
      "Version":"2012-10-17",
      "Statement":
      {
         "Effect":"Allow",
         "Action":"ec2:Describe*",
         "Resource":"*"
      }
   }
   {
     "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
     }
   }
```

You can, however, meet the intention of the previous example with the use of correct policy grammar. Instead of including two complete policy objects each with its own `Statement` element,

you can combine the two blocks into a single `Statement` element. The `Statement` element has an array of two objects as its value, as shown in the following example (called out in **bold**):

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "ec2:Describe*",
          "Resource":"*"
        },
        {
          "Effect": "Allow",
          "Action": "s3:*",
          "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
      ]
    }
```

More than one JSON statement element

This error might at first appear to be a variation on the previous section. However, syntactically it is a different type of error. The following example has only one policy object as denoted by a single pair of { } braces at the top level. However, that object contains two `Statement` elements within it.

An IAM policy must contain only one `Statement` element, consisting of the name (`Statement`) appearing to the left of a colon, followed by its value on the right. The value of a `Statement` element must be an object, denoted by { } braces, containing one `Effect` element, one `Action` element, and one `Resource` element. The following example is incorrect because it contains two `Statement` elements in the policy object (called out in *red*):

JSON

```
{
      "Version":"2012-10-17",
      "Statement": {
        "Effect": "Allow",
```

```
        "Action": "ec2:Describe*",
        "Resource": "*"
      },
      "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
      }
    }
```

A value object can be an array of multiple value objects. To solve this problem, combine the two `Statement` elements into one element with an object array, as shown in the following example (called out in **bold**):

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "ec2:Describe*",
          "Resource":"*"
        },
        {
          "Effect": "Allow",
          "Action": "s3:*",
          "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
     ]
    }
```

The value of the `Statement` element is an object array. The array in this example consists of two objects, each of which is by itself is a correct value for a `Statement` element. Each object in the array is separated by commas.

More than one effect, action, or resource element in a JSON statement element

On the value side of the `Statement` name/value pair, the object must consist of only one `Effect` element, one `Action` element, and one `Resource` element. The following policy is incorrect because it has two `Effect` elements in the `Statement`:

JSON

```
{
      "Version":"2012-10-17",
      "Statement": {
        "Effect": "Deny",
        "Effect": "Allow",
        "Action": "ec2:* ",
        "Resource": "*"
      }
    }
```

Note

The policy engine does not allow such errors in new or edited policies. However, the policy engine continues to permit policies that were saved before the engine was updated. The behavior of existing policies with the error is as follows:

- Multiple `Effect` elements: only the last `Effect` element is observed. The others are ignored.
- Multiple `Action` elements: all `Action` elements are combined internally and treated as if they are a single list.
- Multiple `Resource` elements: all `Resource` elements are combined internally and treated as if they are a single list.

The policy engine does not allow you to save any policy with syntax errors. Correct the errors in the policy before saving. Review and correct any policy validation recommendations for your policies.

In each case, the solution is to remove the incorrect extra element. For `Effect` elements, this is straightforward: if you want the previous example to *deny* permissions to Amazon EC2 instances, then you must remove the line `"Effect": "Allow",` from the policy, as follows:

JSON

```
{
      "Version":"2012-10-17",
      "Statement": {
        "Effect": "Deny",
        "Action": "ec2:*",
        "Resource": "*"
      }
    }
```

However, if the duplicate element is `Action` or `Resource`, then the resolution can be more complicated. You might have multiple actions that you want to allow (or deny) permission to, or you might want to control access to multiple resources. For example, the following example is incorrect because it has multiple `Resource` elements (called out in *red*):

JSON

```
{
      "Version":"2012-10-17",
      "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
      }
    }
```

Each of the required elements in a `Statement` element's value object can be present only once. The solution is to place each value in an array. The following example illustrates this by making the two separate resource elements into one `Resource` element with an array as the value object (called out in **bold**):

JSON

```
{
      "Version":"2012-10-17",
      "Statement": {
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket",
          "arn:aws:s3:::amzn-s3-demo-bucket/*"
        ]
      }
    }
```

Missing JSON version element

A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. In comparison, a policy version is created when you change a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. To learn more about the `Version` policy element see IAM JSON policy elements: Version. To learn more about policy versions, see the section called “Versioning IAM policies”.

As AWS features evolve, new capabilities are added to IAM policies to support those features. Sometimes, an update to the policy syntax includes a new version number. If you use newer features of the policy grammar in your policy, then you must tell the policy parsing engine which version you are using.

The default policy version is "2008-10-17." If you want to use any policy feature that was introduced later, then you must specify the version number that supports the feature you want. We recommend that you *always* include the latest policy syntax version number, which is currently `"Version": "2012-10-17"`.

For example, the following policy is incorrect because it uses a policy variable `${...}` in the ARN for a resource. But it fails to specify a policy syntax version that supports policy variables (called out in *red*):

```
{
```

```
  "Statement":
  {
    "Action": "iam:*AccessKey*",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::123456789012:user/${aws:username}"
  }
}
```

Adding a `Version` element at the top of the policy with the value `2012-10-17`, the first IAM API version that supports policy variables, solves this problem (called out in **bold**):

JSON

```
{
  "Version":"2012-10-17",
  "Statement":
  {
    "Action": "iam:*AccessKey*",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::123456789012:user/${aws:username}"
  }
}
```

Troubleshoot Passkeys and FIDO Security Keys

Use the information here to help you diagnose common issues that you might encounter when working with FIDO2 security keys.

Topics

- I can't enable my FIDO security key
- I can't sign in using my FIDO security key
- I lost or broke my FIDO security key
- Other issues

I can't enable my FIDO security key

Consult the following solutions depending on your status as an IAM user or system administrator

IAM users

If you can't enable your FIDO security key, check the following:

- Are you using a supported configuration?

  IAM supports FIDO2 security devices that connect to your devices through USB, Bluetooth, or NFC. IAM also supports platform authenticators such as TouchID or FaceID. IAM does not support local passkey registration for Windows Hello. To create and use passkeys, Windows users should use cross-device authentication where you use a passkey from one device like a mobile device or hardware security key to sign in on another device like a laptop.

  For information on devices and browsers you can use with WebAuthn and AWS, see Supported configurations for using passkeys and security keys.
- Are you using any browser plugins?

  AWS does not support the use of plugins to add WebAuthn browser support. Instead, use a browser that offers native support of the WebAuthn standard.

  Even if you're using a supported browser, you may have a plugin that is incompatible with WebAuthn. An incompatible plugin may prevent you from enabling and using your FIDO-compliant security key. Disable any plugins that might be incompatible and restart your browser. Then, retry enabling the FIDO security key.
- Do you have the appropriate permissions?

  If you don't have any of the above compatibility issues, you may not have the appropriate permissions. Contact your system administrator.

System administrators

If your IAM users can't enable their FIDO security keys despite using a supported configuration, check their permissions. For a detailed example, see IAM tutorial: Permit users to manage their credentials and MFA settings.

I can't sign in using my FIDO security key

If you can't sign in to the AWS Management Console using your FIDO security key, first see Supported configurations for using passkeys and security keys. If you're using a supported configuration but cannot sign in, contact your system administrator for assistance.

I lost or broke my FIDO security key

Up to **eight** MFA devices of any combination of the currently supported MFA types can be assigned to a user. With multiple MFA devices, you only need one MFA device to sign in to the AWS Management Console. Replacing a FIDO security key is similar to replacing a hardware TOTP token. If you lose or break any type of MFA device, see Recover an MFA protected identity in IAM.

Other issues

If you have an issue with FIDO security keys that is not covered here, do one of the following:

- IAM users: Contact your system administrator.
- AWS account root users: Contact AWS Support.

Troubleshoot IAM roles

Use the information here to help you diagnose and fix common issues that you might encounter when working with IAM roles.

Topics

- I can't assume a role
- A new role appeared in my AWS account
- I can't edit or delete a role in my AWS account
- I'm not authorized to perform: iam:PassRole
- Why can't I assume a role with a 12-hour session? (AWS CLI, AWS API)
- I receive an error when I try to switch roles in the IAM console
- My role has a policy that allows me to perform an action, but I get "access denied"
- The service did not create the role's default policy version
- There is no use case for a service role in the console

I can't assume a role

Check the following:

- To allow users to assume the current role again within a role session, specify the role ARN or AWS account ARN as a principal in the role trust policy. AWS services that provide compute

resources such as Amazon EC2, Amazon ECS, Amazon EKS, and Lambda provide temporary credentials and automatically update these credentials. This ensures that you always have a valid set of credentials. For these services, it's not necessary to assume the current role again to obtain temporary credentials. However, if you intend to pass session tags or a session policy, you need to assume the current role again. To learn how to modify a role trust policy to add the principal role ARN or AWS account ARN, see Update a role trust policy.

- When you assume a role using the AWS Management Console, make sure to use the exact name of your role. Role names are case sensitive when you assume a role.
- When you assume a role using AWS STS API or AWS CLI, make sure to use the exact name of your role in the ARN. Role names are case sensitive when you assume a role.
- When you assume a role using a SAML-based federated identity provider and SAML encryption is enabled, make sure you have uploaded a valid private decryption key for the SAML identity provider. For more information, see Manage SAML encryption keys.
- Verify that your IAM policy grants you permission to call `sts:AssumeRole` for the role that you want to assume. The `Action` element of your IAM policy must allow you to call the `AssumeRole` action. In addition, the `Resource` element of your IAM policy must specify the role that you want to assume. For example, the `Resource` element can specify a role by its Amazon Resource Name (ARN) or by a wildcard (*). For example, at least one policy applicable to you must grant permissions similar to the following:

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::account_id_number:role/role-name-you-want-to-assume"
```

- Verify that your IAM identity is tagged with any tags that the IAM policy requires. For example, in the following policy permissions, the `Condition` element requires that you, as the principal requesting to assume the role, must have a specific tag. You must be tagged with `department = HR` or `department = CS`. Otherwise, you cannot assume the role. To learn about tagging IAM users and roles, see the section called “Tags for IAM resources”.

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "*",
"Condition": {"StringEquals": {"aws:PrincipalTag/department": [
        "HR",
        "CS"
    ]}}
```

- Verify that you meet all the conditions that are specified in the role's trust policy. A `Condition` can specify an expiration date, an external ID, or that a request must come only from specific IP addresses. Consider the following example: If the current date is any time after the specified date, then the policy never matches and cannot grant you the permission to assume the role.

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::account_id_number:role/role-name-you-want-to-assume"
"Condition": {
    "DateLessThan" : {
        "aws:CurrentTime" : "2016-05-01T12:00:00Z"
    }
}
```

- Verify that the AWS account from which you are calling `AssumeRole` is a trusted entity for the role that you are assuming. Trusted entities are defined as a `Principal` in a role's trust policy. The following example is a trust policy that is attached to the role that you want to assume. In this example, the account ID with the IAM user that you signed in with must be 123456789012. If your account number is not listed in the `Principal` element of the role's trust policy, then you cannot assume the role. It does not matter what permissions are granted to you in access policies. Note that the example policy limits permissions to actions that occur between July 1, 2017 and December 31, 2017 (UTC), inclusive. If you log in before or after those dates, then the policy does not match, and you cannot assume the role.

```
"Effect": "Allow",
"Principal": { "AWS": "arn:aws:iam::123456789012:root" },
"Action": "sts:AssumeRole",
"Condition": {
  "DateGreaterThan": {"aws:CurrentTime": "2017-07-01T00:00:00Z"},
  "DateLessThan": {"aws:CurrentTime": "2017-12-31T23:59:59Z"}
}
```

- **Source Identity** – Administrators can configure roles to require identities to pass a custom string that identifies the person or application that is performing actions in AWS, called *source identity*. Verify whether the role being assumed requires that a source identity is set. For more information about source identity, see Monitor and control actions taken with assumed roles.

A new role appeared in my AWS account

Some AWS services require that you use a unique type of service role that is linked directly to the service. This service-linked role is predefined by the service and includes all the permissions that the service requires. This makes setting up a service easier because you don't have to manually add the necessary permissions. For general information about service-linked roles, see Create a service-linked role.

You might already be using a service when it begins supporting service-linked roles. If so, you might receive an email telling you about a new role in your account. This role includes all the permissions that the service needs to perform actions on your behalf. You don't need to take any action to support this role. However, you should not delete the role from your account. Doing so could remove permissions that the service needs to access AWS resources. You can view the service-linked roles in your account by going to the IAM **Roles** page of the IAM console. Service-linked roles appear with **(Service-linked role)** in the **Trusted entities** column of the table.

For information about which services support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. For information about using the service-linked role for a service, choose the **Yes** link.

I can't edit or delete a role in my AWS account

You cannot delete or edit the permissions for a service-linked role in IAM. These roles include predefined trusts and permissions that are required by the service in order to perform actions on your behalf. You can use the IAM console, AWS CLI, or API to edit only the description of a service-linked role. You can view the service-linked roles in your account by going to the IAM **Roles** page in the console. Service-linked roles appear with **(Service-linked role)** in the **Trusted entities** column of the table. A banner on the role's **Summary** page also indicates that the role is a service-linked role. You can manage and delete these roles only through the linked service, if that service supports the action. Be careful when modifying or deleting a service-linked role because doing so could remove permissions that the service needs to access AWS resources.

For information about which services support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column.

I'm not authorized to perform: iam:PassRole

When you create a service-linked role, you must have permission to pass that role to the service. Some services automatically create a service-linked role in your account when

you perform an action in that service. For example, Amazon EC2 Auto Scaling creates the `AWSServiceRoleForAutoScaling` service-linked role for you the first time that you create an Auto Scaling group. If you try to create an Auto Scaling group without the `PassRole` permission, you receive the following error:

```
ClientError: An error occurred (AccessDenied) when calling the
PutLifecycleHook operation: User: arn:aws:sts::111122223333:assumed-role/
Testrole/Diego is not authorized to perform: iam:PassRole on resource:
arn:aws:iam::111122223333:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling
```

To fix this error, ask your administrator to add the `iam:PassRole` permission for you.

To learn which services support service-linked roles, see AWS services that work with IAM. To learn whether a service automatically creates a service-linked role for you, choose the **Yes** link to view the service-linked role documentation for the service.

Why can't I assume a role with a 12-hour session? (AWS CLI, AWS API)

When you use the AWS STS `AssumeRole*` API or `assume-role*` CLI operations to assume a role, you can specify a value for the `DurationSeconds` parameter. You can specify a value from 900 seconds (15 minutes) up to the **Maximum session duration** setting for the role. If you specify a value higher than this setting, the operation fails. This setting can have a maximum value of 12 hours. For example, if you specify a session duration of 12 hours, but your administrator set the maximum session duration to 6 hours, your operation fails. To learn how to view the maximum value for your role, see Update the maximum session duration for a role.

If you use *role chaining* (using a role to assume a second role), your session is limited to a maximum of one hour. If you then use the `DurationSeconds` parameter to provide a value greater than one hour, the operation fails.

I receive an error when I try to switch roles in the IAM console

The information you enter on the **Switch Role** page must match the information for the role. Otherwise, the operation fails and you receive the following error:

```
Invalid information in one or more fields. Check your information or
contact your administrator.
```

If you receive this error, confirm that the following information is correct:

- **Account ID or alias** – The AWS account ID is a 12-digit number. Your account might have an alias, which is a friendly identifier such as your company name that can be used instead of your AWS account ID. You can use either the account ID or the alias in this field.
- **Role name** – Role names are case sensitive. The account ID and role name must match what is configured for the role.

If you continue to receive an error message, contact your administrator to verify the previous information. The role trust policy or the IAM user policy might limit your access. Your administrator can verify the permissions for these policies.

My role has a policy that allows me to perform an action, but I get "access denied"

Your role session might be limited by session policies. When you request temporary security credentials programmatically using AWS STS, you can optionally pass inline or managed session policies. Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary credential session for a role. You can pass a single JSON inline session policy document using the `Policy` parameter. You can use the `PolicyArns` parameter to specify up to 10 managed session policies. The resulting session's permissions are the intersection of the role's identity-based policies and the session policies. Alternatively, if your administrator or a custom program provides you with temporary credentials, they might have included a session policy to limit your access.

The service did not create the role's default policy version

A service role is a role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. In some cases, the service creates the service role and its policy in IAM for you. Although you can modify or delete the service role and its policy from within IAM, AWS does not recommend this. The role and policy are intended for use only by that service. If you edit the policy and set up another environment, when the service tries to use the same role and policy, the operation can fail.

For example, when you use AWS CodeBuild for the first time, the service creates a role named `codebuild-RWBCore-service-role`. That service role uses the policy named `codebuild-RWBCore-managed-policy`. If you edit the policy, it creates a new version and saves that version

as the default version. If you perform a subsequent operation in AWS CodeBuild, the service might try to update the policy. If it does, you receive the following error:

```
codebuild.amazon.com did not create the default version (V2) of the
codebuild-RWBCore-managed-policy policy that is attached to the codebuild-
RWBCore-service-role role. To continue, detach the policy from any other
identities and then delete the policy and the role.
```

If you receive this error, you must make changes in IAM before you can continue with your service operation. First, set the default policy version to V1 and try the operation again. If V1 was previously deleted, or if choosing V1 doesn't work, then clean up and delete the existing policy and role.

For more information on editing managed policies, see Editing customer managed policies (console). For more information about policy versions, see Versioning IAM policies.

To delete a service role and its policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. In the list of policies, choose the name of the policy that you want to delete.
4. Choose the **Entities attached** tab to view which IAM users, groups, or roles use this policy. If any of these identities use the policy, complete the following tasks:
    a. Create a new managed policy with the necessary permissions. To ensure that the identities have the same permissions before and after your actions, copy the JSON policy document from the existing policy. Then create the new managed policy and paste the JSON document as described in Creating Policies using the JSON editor.
    b. For each affected identity, attach the new policy and then detach the old one. For more information, see Adding and removing IAM identity permissions.
5. In the navigation pane, choose **Roles**.
6. In the list of roles, choose the name of the role that you want to delete.
7. Choose the **Trust relationships** tab to view which entities can assume the role. If any entity other than the service is listed, complete the following tasks:
    a. Create a new role that trusts those entities.

   b. The policy that you created in the previous step. If you skipped that step, create the new managed policy now.
   c. Notify anyone who was assuming the role that they can no longer do so. Provide them with information about how to assume the new role and have the same permissions.
8. Delete the policy.
9. Delete the role.

There is no use case for a service role in the console

Some services require that you manually create a service role to grant the service permissions to perform actions on your behalf. If the service is not listed in the IAM console, you must manually list the service as the trusted principal. If the documentation for the service or feature that you are using does not include instructions for listing the service as the trusted principal, provide feedback for the page.

To manually create a service role, you must know the service principal for the service that will assume the role. A service principal is an identifier that is used to grant permissions to a service. The service principal is defined by the service.

You can find the service principal for some services by checking the following:

1. Open AWS services that work with IAM.
2. Check whether the service has **Yes** in the **Service-linked roles** column.
3. Choose the **Yes** link to view the service-linked role documentation for that service.
4. Find the Service-linked role permissions section for that service to view the service principal.

You can manually create a service role using AWS CLI commands or AWS API operations. To manually create a service role using the IAM console, complete the following tasks:

1. Create an IAM role using your account ID. Do not attach a policy or grant any permissions. For details, see Create a role to give permissions to an IAM user.
2. Open the role and edit the trust relationship. Instead of trusting the account, the role must trust the service. For example, update the following `Principal` element:

```
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

Change the principal to the value for your service, such as IAM.

```
"Principal": { "Service": "iam.amazonaws.com" }
```

3. Add the permissions that the service requires by attaching permissions policies to the role.
4. Return to the service that requires the permissions and use the documented method to notify the service about the new service role.

Troubleshoot IAM and Amazon EC2

The following information can help you troubleshoot IAM issues with Amazon EC2.

Topics

- When I try to launch an instance, I don't see the role in the Amazon EC2 console IAM Role list
- The credentials on my instance are for the wrong role
- When I attempt to call the AddRoleToInstanceProfile, I get an AccessDenied error
- Amazon EC2: When I try to launch an instance with a role, I get an AccessDenied error
- I can't access the temporary security credentials on my EC2 instance
- What do the errors from the info document in the IAM subtree mean?

When I try to launch an instance, I don't see the role in the Amazon EC2 console IAM Role list

Check the following:

- If you are signed in as an IAM user, verify that you have permission to call `ListInstanceProfiles`. For information about the permissions necessary to work with roles, see Permissions required for using roles with Amazon EC2. For information about adding permissions to a user, see Manage IAM policies.

  If you cannot modify your own permissions, you must contact an administrator who can work with IAM in order to update your permissions.
- If you created a role using the IAM CLI or API, verify the following:
  - You created an instance profile and added the role to that instance profile.

- You used the same name for the role and the instance profile. If you name your role and instance profile differently, you won't see the correct role name in the Amazon EC2 console.

The **IAM Role** list in the Amazon EC2 console lists the names of instance profiles, not the names of roles. You will have to select the name of the instance profile that contains the role you want. For details about instance profiles, see Use instance profiles.

> **Note**
>
> If you use the IAM console to create roles, you don't need to work with instance profiles. For each role that you create in the IAM console, an instance profile is created with the same name as the role, and the role is automatically added to that instance profile. An instance profile can contain only one IAM role, and that limit cannot be increased.

The credentials on my instance are for the wrong role

The role in the instance profile might have been replaced recently. If so, your application will need to wait for the next automatically scheduled credential rotation before credentials for your role become available.

To force the change, you must disassociate the instance profile and then associate the instance profile, or you can stop your instance and then restart it.

When I attempt to call the `AddRoleToInstanceProfile`, I get an `AccessDenied` error

If you are making requests as an IAM user, verify that you have the following permissions:

- `iam:AddRoleToInstanceProfile` with the resource matching the instance profile ARN (for example, `arn:aws:iam::999999999999:instance-profile/ExampleInstanceProfile`).

For more information about the permissions necessary to work with roles, see How do I get started?. For information about adding permissions to a user, see Manage IAM policies.

Amazon EC2: When I try to launch an instance with a role, I get an `AccessDenied` error

Check the following:

- Launch an instance without an instance profile. This will help ensure that the problem is limited to IAM roles for Amazon EC2 instances.
- If you are making requests as an IAM user, verify that you have the following permissions:
  - `ec2:RunInstances` with a wildcard resource ("*")
  - `iam:PassRole` with the resource matching the role ARN (for example, `arn:aws:iam::999999999999:role/ExampleRoleName`)
- Call the IAM `GetInstanceProfile` action to ensure that you are using a valid instance profile name or a valid instance profile ARN. For more information, see Using IAM roles with Amazon EC2 instances.
- Call the IAM `GetInstanceProfile` action to ensure that the instance profile has a role. Empty instance profiles will fail with an `AccessDenied` error. For more information about creating a role, see IAM role creation.

For more information about the permissions necessary to work with roles, see How do I get started?. For information about adding permissions to a user, see Manage IAM policies.

I can't access the temporary security credentials on my EC2 instance

To access temporary security credentials on your EC2 instance, you must first use the IAM console to create a role. Then you launch an EC2 instance that uses that role and examine the running instance. For more information, see **How Do I Get Started?** in Use an IAM role to grant permissions to applications running on Amazon EC2 instances.

If you still can't access your temporary security credentials on your EC2 instance, check the following:

- Can you access another part of the Instance Metadata Service (IMDS)? If not, check that you have no firewall rules blocking access to requests to the IMDS.

```
[ec2-user@domU-12-31-39-0A-8D-DE ~]$ GET http://169.254.169.254/latest/meta-data/
hostname; echo
```

- Does the `iam` subtree of the IMDS exist? If not, verify that your instance has an IAM instance profile associated with it by calling the EC2 `DescribeInstances` API operation or using the `aws ec2 describe-instances` CLI command.

```
[ec2-user@domU-12-31-39-0A-8D-DE ~]$ GET http://169.254.169.254/latest/meta-data/iam;
 echo
```

- Check the `info` document in the IAM subtree for an error. If you have an error, see What do the errors from the `info` document in the IAM subtree mean? for more information.

```
[ec2-user@domU-12-31-39-0A-8D-DE ~]$ GET http://169.254.169.254/latest/meta-data/iam/
info; echo
```

What do the errors from the `info` document in the IAM subtree mean?

The `iam/info` document indicates `"Code":"InstanceProfileNotFound"`

Your IAM instance profile has been deleted and Amazon EC2 can no longer provide credentials to your instance. You must attach a valid instance profile to your Amazon EC2 instance.

If an instance profile with that name exists, check that the instance profile wasn't deleted and another was created with the same name:

1. Call the IAM `GetInstanceProfile` operation to get the `InstanceProfileId`.
2. Call the Amazon EC2 `DescribeInstances` operation to get the `IamInstanceProfileId` for the instance.
3. Verify that the `InstanceProfileId` from the IAM operation matches the `IamInstanceProfileId` from the Amazon EC2 operation.

If the IDs are different, then the instance profile attached to your instances is no longer valid. You must attach a valid instance profile to the instance.

The `iam/info` document indicates a success but indicates `"Message":"Instance Profile does not contain a role..."`

The role has been removed from the instance profile by the IAM `RemoveRoleFromInstanceProfile` action. You can use the IAM `AddRoleToInstanceProfile`

action to attach a role to the instance profile. Your application will need to wait until the next scheduled refresh to access the credentials for the role.

To force the change, you must disassociate the instance profile and then associate the instance profile, or you can stop your instance and then restart it.

The `iam/security-credentials/[role-name]` document indicates `"Code":"AssumeRoleUnauthorizedAccess"`

Amazon EC2 does not have permission to assume the role. Permission to assume the role is controlled by the trust policy attached to the role, like the example that follows. Use the IAM `UpdateAssumeRolePolicy` API to update the trust policy.

JSON

```
{"Version":"2012-10-17",       "Statement": [{"Effect": "Allow","Principal":
 {"Service": ["ec2.amazonaws.com"]},"Action": ["sts:AssumeRole"]}]}
```

Your application will need to wait until the next automatically scheduled refresh to access the credentials for the role.

To force the change, you must disassociate the instance profile and then associate the instance profile, or you can stop your instance and then restart it.

Troubleshoot IAM and Amazon S3

Use the information here to help you troubleshoot and fix issues that you might encounter when working with Amazon S3 and IAM.

How do I grant anonymous access to an Amazon S3 bucket?

You use an Amazon S3 bucket policy that specifies a wildcard (*) in the `principal` element, which means anyone can access the bucket. With anonymous access, anyone (including users without an AWS account) will be able to access the bucket. For a sample policy, see Example Cases for Amazon S3 Bucket Policies in the *Amazon Simple Storage Service User Guide*.

I'm signed in as an AWS account root user. Why can't I access an Amazon S3 bucket under my account?

In some cases, you might have an IAM user with full access to IAM and Amazon S3. If the IAM user assigns a bucket policy to an Amazon S3 bucket and doesn't specify the root user as a principal, the root user is denied access to that bucket. However, as the root user, you can still access the bucket. To do that, modify the bucket policy to allow root user access from the Amazon S3 console or the AWS CLI. Use the following principal, replacing *123456789012* with the ID of the AWS account.

```
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

Troubleshoot SAML federation with IAM

Use the information here to help you diagnose and fix issues that you might encounter when working with SAML 2.0 and federation with AWS Identity and Access Management.

Topics

- Error: Your request included an invalid SAML response. To logout, click here.
- Error: RoleSessionName is required in AuthnResponse (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)
- Error: Not authorized to perform sts:AssumeRoleWithSAML (service: AWSSecurityTokenService; status code: 403; error code: AccessDenied)
- Error: RoleSessionName in AuthnResponse must match [a-zA-Z_0-9+=,.@-]{2,64} (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)
- Error: Source Identity must match [a-zA-Z_0-9+=,.@-]{2,256} and not begin with "aws:" (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)
- Error: Response signature invalid (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)
- Error: Invalid private key.
- Error: Failed to remove private key.
- Error: Failed to remove private key because the Key ID does not match a private key.
- Error: Failed to assume role: Issuer not present in specified provider (service: AWSOpenIdDiscoveryService; status code: 400; error code: AuthSamlInvalidSamlResponseException)

- Error: Could not parse metadata.
- Error: Unable to update identity provider. No updates are defined for metadata or encryption assertion.
- Error: Unable to set assertion encryption mode to Required because no private key is provided.
- Error: Unable to add and remove private keys in the same request. Set a value for only one of the two parameters.
- Error: Specified provider doesn't exist.
- Error: Requested DurationSeconds exceeds MaxSessionDuration set for this role.
- Error: Private key limit of 2 is reached.
- Error: Response does not contain the required audience.

Error: Your request included an invalid SAML response. To logout, click here.

This error can occur when the SAML response from the identity provider does not include an attribute with the `Name` set to `https://aws.amazon.com/SAML/Attributes/Role`. The attribute must contain one or more `AttributeValue` elements, each containing a comma-separated pair of strings:

- The ARN of a role that the user can be mapped to
- The ARN of the SAML provider

The error can also occur when the SAML attribute values sent by the identity provider have either a leading or trailing whitespace, or other invalid characters in the SAML attribute values. For more information on the expected values for SAML attributes, see Configure SAML assertions for the authentication response

For more information, see Configure SAML assertions for the authentication response. To view the SAML response in your browser, follow the steps listed in View a SAML response in your browser.

Error: RoleSessionName is required in AuthnResponse (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur when the SAML response from the identity provider does not include an attribute with the `Name` set to `https://aws.amazon.com/SAML/Attributes/RoleSessionName`. The attribute value is an identifier for the user and is typically a user ID or an email address.

For more information, see Configure SAML assertions for the authentication response. To view the SAML response in your browser, follow the steps listed in View a SAML response in your browser.

Error: Not authorized to perform sts:AssumeRoleWithSAML (service: AWSSecurityTokenService; status code: 403; error code: AccessDenied)

This error can occur if the IAM role specified in the SAML response is misspelled or does not exist. Make sure to use the exact name of your role, because role names are case sensitive. Correct the name of the role in the SAML service provider configuration.

You are allowed access only if your role trust policy includes the `sts:AssumeRoleWithSAML` action. If your SAML assertion is configured to use the `PrincipalTag` attribute, your trust policy must also include the `sts:TagSession` action. For more information about session tags, see Pass session tags in AWS STS.

This error can occur if you do not have `sts:SetSourceIdentity` permissions in your role trust policy. If your SAML assertion is configured to use the `SourceIdentity` attribute, then your trust policy must also include the `sts:SetSourceIdentity` action. For more information about source identity, see Monitor and control actions taken with assumed roles.

This error can also occur if a federated principal does not have permissions to assume the role. The role must have a trust policy that specifies the ARN of the IAM SAML identity provider as the `Principal`. The role also contains conditions that control which users can assume the role. Ensure that your users meet the requirements of the conditions.

This error can also occur if the SAML response does not include a `Subject` containing a `NameID`.

For more information, see Enabling SAML 2.0 federated principals to access the AWS Management Console and Configure SAML assertions for the authentication response. To view the SAML response in your browser, follow the steps listed in View a SAML response in your browser.

Error: RoleSessionName in AuthnResponse must match [a-zA-Z_0-9+=,.@-]{2,64} (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur if the `RoleSessionName` attribute value is too long or contains invalid characters. The maximum valid length is 64 characters.

For more information, see Configure SAML assertions for the authentication response. To view the SAML response in your browser, follow the steps listed in View a SAML response in your browser.

Error: Source Identity must match [a-zA-Z_0-9+=,.@-]{2,256} and not begin with "aws:" (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur if the `sourceIdentity` attribute value is too long or contains invalid characters. The maximum valid length is 256 characters. For more information about source identity, see Monitor and control actions taken with assumed roles.

For more information about creating SAML assertions, see Configure SAML assertions for the authentication response. To view the SAML response in your browser, follow the steps listed in View a SAML response in your browser.

Error: Response signature invalid (service: AWSSecurityTokenService; status code: 400; error code: InvalidIdentityToken)

This error can occur when federation metadata of the identity provider does not match the metadata of the IAM identity provider. For example, the metadata file for the identity service provider might have changed to update an expired certificate. Download the updated SAML metadata file from your identity service provider. Then update it in the AWS identity provider entity that you define in IAM with the `aws iam update-saml-provider` cross-platform CLI command or the `Update-IAMSAMLProvider` PowerShell cmdlet.

Error: Invalid private key.

This error can occur if you do not format your private key file properly. This error may provide additional detail about why the private key is invalid:

- Key is encrypted.

- Key format is not recognized. Private key file must be a .pem file.

When you Create a SAML identity provider in IAM in the AWS Management Console, you must download the private key from your identity provider to provide to IAM to enable encryption. The private key must be a .pem file that uses AES-GCM or AES-CBC encryption algorithm to decrypt SAML assertions.

Error: Failed to remove private key.

This error can occur when SAML encryption is set to Required, and your request would remove the only private decryption key for the IAM SAML provider. For more information about rotating private keys, see Manage SAML encryption keys.

Error: Failed to remove private key because the Key ID does not match a private key.

This error can occur if the `keyId` value for the private key does not match either Key ID for the identity provider's private key files.

When you use update-saml-provider or UpdateSAMLProvider API operations to remove SAML encryption private keys, the value in `RemovePrivateKey` must be a valid Key ID for a private key attached to your identity provider.

Error: Failed to assume role: Issuer not present in specified provider (service: AWSOpenIdDiscoveryService; status code: 400; error code: AuthSamlInvalidSamlResponseException)

This error can occur if the issuer in the SAML response does not match the issuer declared in the federation metadata file. The metadata file was uploaded to AWS when you created the identity provider in IAM.

Error: Could not parse metadata.

This error can occur if you do not format your metadata file properly.

When you create or manage a SAML identity provider in the AWS Management Console, you must retrieve the SAML metadata document from your identity provider.

This metadata file includes the issuer name, expiration information, and keys that can be used to validate the SAML authentication response (assertions) received from the IdP. The metadata file must be encoded in UTF-8 format without a byte order mark (BOM). To remove the BOM, you can encode the file as UTF-8 using a text editing tool, such as Notepad++.

The X.509 certificate included as part of the SAML metadata document must use a key size of at least 1024 bits. Also, the X.509 certificate must also be free of any repeated extensions. You can use extensions, but the extensions can only appear once in the certificate. If the X.509 certificate does not meet either condition, IdP creation fails and returns an "Unable to parse metadata" error.

As defined by the SAML V2.0 Metadata Interoperability Profile Version 1.0, IAM does not evaluate or take action on the expiration of X.509 certificates in SAML metadata documents. If you are concerned about expired X.509 certificates, we recommend monitoring certificate expiration dates and rotating certificates according to your organization's governance and security policies.

Error: Unable to update identity provider. No updates are defined for metadata or encryption assertion.

This error can occur if you use the `update-saml-provider` CLI or `UpdateSAMLProvider` API operations, but don't provide update values in your request parameters. For more information on updating your IAM SAML provider, see Create a SAML identity provider in IAM.

Error: Unable to set assertion encryption mode to Required because no private key is provided.

This error can occur when you haven't previously uploaded a private decryption key and you attempt to set SAML encryption to Required without including a private key in your request.

Make sure that a private key is defined for your IAM SAML provider when using `create-saml-provider` CLI, `CreateSAMLProvider` API, `update-saml-provider` CLI, or `UpdateSAMLProvider` API operations to require encrypted SAML assertions.

Error: Unable to add and remove private keys in the same request. Set a value for only one of the two parameters.

This error can occur if both add and remove private key values are included in the same request.

When you use update-saml-provider or UpdateSAMLProvider API operations to rotate SAML encryption private key files, you can only add or remove a private key in your request. If you add a

private key while removing a private key, the operation fails. For more information about rotating private keys, see Manage SAML encryption keys.

Error: Specified provider doesn't exist.

This error can occur if the name of the provider in the SAML assertion does not match the name of the provider in IAM. For more information about viewing the provider name, see Create a SAML identity provider in IAM.

Error: Requested DurationSeconds exceeds MaxSessionDuration set for this role.

This error can occur if you assume a role from the AWS CLI or API.

When you use the assume-role-with-saml CLI or AssumeRoleWithSAML API operations to assume a role, you can specify a value for the `DurationSeconds` parameter. You can specify a value from 900 seconds (15 minutes) up to the maximum session duration setting for the role. If you specify a value higher than this setting, the operation fails. For example, if you specify a session duration of 12 hours, but your administrator set the maximum session duration to 6 hours, your operation fails. To learn how to view the maximum value for your role, see Update the maximum session duration for a role.

Error: Private key limit of 2 is reached.

This error can occur if you try to add a private key to your identity provider.

You can save up to two private keys for each identity provider. When you use update-saml-provider or UpdateSAMLProvider API operations to add a third private key, the operation fails.

Remove private keys that have expired before adding a new private key. For more information about rotating private keys, see Manage SAML encryption keys.

Error: Response does not contain the required audience.

This error can occur if there is a mismatch between the audience URL and the identity provider in the SAML configuration. Make sure that your identity provider (IdP) relying party identifier exactly matches the audience URL (entity ID) provided in the SAML configuration.

How IAM works with other AWS services

While IAM is the primary AWS service you will use to manage IAM resources, all other AWS services work with IAM to control access to the resources in your account.

- CloudFormation

  CloudFormation integrates with IAM by allowing you to define and manage IAM resources as part of your CloudFormation templates. You can use CloudFormation to specify the necessary IAM permissions for other AWS resources you provision. CloudFormation also supports the use of IAM roles to manage the credentials required for provisioning and managing your AWS infrastructure, and its drift detection feature helps you maintain the integrity of your IAM configurations.

- AWS CloudShell

  When you access AWS CloudShell, your authentication and authorization are handled through IAM. AWS CloudShell runs within the context of an IAM role assigned to your user or account. When you launch AWS CloudShell, it automatically generates temporary security credentials based on the IAM role assigned to you.

- AWS SDKs

  The AWS SDKs work with IAM by handling the authentication and authorization process, managing AWS credentials, and respecting the permissions and policies defined in IAM to ensure your application can only access the resources it is authorized to use. The SDKs provide mechanisms for obtaining and using temporary security credentials, as well as validating the permissions required for your application's operations.

For a list of AWS services that work with IAM and the IAM features the services support, see AWS services that work with IAM.

Create IAM resources with CloudFormation

AWS Identity and Access Management is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (such as access keys, groups, group policies, instance profiles, managed policies, OIDC providers, inline policies, roles, role policies, SAML providers, server certificates,

service-linked roles, users (and adding users to groups), user policies, and virtual MFA devices), and CloudFormation provisions and configures those resources for you.

When you use CloudFormation, you can reuse your template to set up your IAM resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

IAM and CloudFormation templates

To provision and configure resources for IAM and related services, you must understand CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use CloudFormation Designer to help you get started with CloudFormation templates. For more information, see What is CloudFormation Designer? in the *AWS CloudFormation User Guide*.

IAM supports creating access keys, groups, group policies, instance profiles, managed policies, OIDC providers, inline policies, roles, role policies, SAML providers, server certificates, service-linked roles, users (and adding users to groups), user policies, and virtual MFA devices in CloudFormation. For more information, including examples of JSON and YAML templates for IAM resources, see the AWS Identity and Access Management resource type reference in the *AWS CloudFormation User Guide*.

You can also create templates that create related resources, such as roles and managed policies.

Learn more about CloudFormation

To learn more about CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- CloudFormation API Reference
- AWS CloudFormation Command Line Interface User Guide

Use AWS CloudShell to work with AWS Identity and Access Management

AWS CloudShell is a browser-based, pre-authenticated shell that you can launch directly from the AWS Management Console. You can run AWS CLI commands against AWS services (including AWS Identity and Access Management) using your preferred shell (Bash, PowerShell or Z shell). And you can do this without needing to download or install command line tools.

You launch AWS CloudShell from the AWS Management Console, and the AWS credentials you used to sign in to the console are automatically available in a new shell session. This pre-authentication of AWS CloudShell users allows you to skip configuring credentials when interacting with AWS services such as IAM using AWS CLI version 2 (pre-installed on the shell's compute environment).

Obtain IAM permissions for AWS CloudShell

Using the access management resources provided by AWS Identity and Access Management, administrators can grant permissions to IAM users so they can access AWS CloudShell and use the environment's features.

The quickest way for an administrator to grant access to users is through an AWS managed policy. An AWS managed policy is a standalone policy that's created and administered by AWS. The following AWS managed policy for CloudShell can be attached to IAM identities:

- `AWSCloudShellFullAccess`: Grants permission to use AWS CloudShell with full access to all features.

If you want to limit the scope of actions that an IAM user can perform with AWS CloudShell, you can create a custom policy that uses the `AWSCloudShellFullAccess` managed policy as a template. For more information about limiting the actions that are available to users in CloudShell, see Managing AWS CloudShell access and usage with IAM policies in the *AWS CloudShell User Guide*.

Interact with IAM

After you launch AWS CloudShell from the AWS Management Console, you can immediately start to interact with IAM using the command line interface.

> **Note**
>
> When using AWS CLI in AWS CloudShell, you don't need to download or install any additional resources. Moreover, because you're already authenticated within the shell, you don't need to configure credentials before making calls.

Create an IAM group and add an IAM user to the group using AWS CloudShell

The following example uses CloudShell to create an IAM group, add an IAM user to the group, and then verify that the command succeeded.

1. From the AWS Management Console, you can launch CloudShell by choosing the following options available on the navigation bar:
   - Choose the CloudShell icon.
   - Start typing "cloudshell" in Search box and then choose the CloudShell option.
2. To create an IAM group, enter the following command in the CloudShell command line. In this example we named the group *east_coast*:

   ```
   aws iam create-group --group-name east_coast
   ```

   If the call is successful, the command line displays a response from the service similar to the following output:

   ```
   {
      "Group": {
          "Path": "/",
          "GroupName": "east_coast",
          "GroupId": "AGPAYBDBW4JBY3EXAMPLE",
          "Arn": "arn:aws:iam::111122223333:group/east_coast",
          "CreateDate": "2023-09-11T21:02:21+00:00"
       }
   }
   ```

3. To add a user to the group that you created, use the following command, specifying the group name and username. In this example we named the group *east_coast* and the user *john*:

```
aws iam add-user-to-group --group-name east_coast --user-name john
```

4. To verify that the user is in the group, use the following command, specifying the group name. In this example we continue using the group *east_coast* :

```
aws iam get-group --group-name east_coast
```

If the call is successful, the command line displays a response from the service similar to the following output:

```
      {
        "Users": [
          {
              "Path": "/",
              "UserName": "john",
              "UserId": "AIDAYBDBW4JBXGEXAMPLE",
              "Arn": "arn:aws:iam::552108220995:user/john",
              "CreateDate": "2023-09-11T20:43:14+00:00",
              "PasswordLastUsed": "2023-09-11T20:59:14+00:00"
          }
        ],
        "Group": {
              "Path": "/",
              "GroupName": "east_coast",
              "GroupId": "AGPAYBDBW4JBY3EXAMPLE",
              "Arn": "arn:aws:iam::111122223333:group/east_coast",
              "CreateDate": "2023-09-11T21:02:21+00:00"
           }
       }
```

Using this service with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	AWS Tools for PowerShell code examples
AWS SDK for Python v4	AWS SDK for Python v4 code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to this service, see Code examples for IAM using AWS SDKs.

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Reference information for AWS Identity and Access Management

Use the topics in this section to find detailed reference material for various aspects of IAM and AWS STS.

Topics

- Identify AWS resources with Amazon Resource Names (ARNs)
- IAM identifiers
- IAM and AWS STS quotas
- Dual-stack endpoint support
- Interface VPC endpoints
- AWS services that work with IAM
- AWS Signature Version 4 for API requests
- IAM JSON policy reference

- Simplified AWS service information for programmatic access

Identify AWS resources with Amazon Resource Names (ARNs)

Amazon Resource Names (ARNs) uniquely identify AWS resources. We require an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies, Amazon Relational Database Service (Amazon RDS) tags, and API calls. While ARNs, like any identifying information, should be used and shared carefully, they are not considered secret, sensitive, or confidential information.

ARN format

The following are the general formats for ARNs. The specific formats depend on the resource. To use an ARN, replace the *italicized* text with the resource-specific information. Be aware that the ARNs for some resources omit the Region, the account ID, or both the Region and the account ID.

```
arn:partition:service:region:account-id:resource-id
```

```
arn:partition:service:region:account-id:resource-type/resource-id
arn:partition:service:region:account-id:resource-type:resource-id
```

partition

The partition in which the resource is located. A *partition* is a group of AWS Regions. Each AWS account is scoped to one partition.

The following are the supported partitions:

- aws - AWS Regions
- aws-cn - China Regions
- aws-us-gov - AWS GovCloud (US) Regions

service

The service namespace that identifies the AWS product.

region

The Region code. For example, us-east-2 for US East (Ohio). For the list of Region codes, see Regional endpoints in the *AWS General Reference*.

account-id

The ID of the AWS account that owns the resource, without the hyphens. For example, 123456789012.

resource-type

The resource type. For example, vpc for a virtual private cloud (VPC).

resource-id

The resource identifier. This is the name of the resource, the ID of the resource, or a resource path. Some resource identifiers include a parent resource (sub-resource-type/parent-resource/sub-resource) or a qualifier such as a version (resource-type:resource-name:qualifier).

Examples

IAM user

arn:aws:iam::*123456789012*:user/*john*

SNS topic

arn:aws:sns:*us-east-1*:*123456789012*:*example-sns-topic-name*

VPC

arn:aws:ec2:*us-east-1*:*123456789012*:vpc/*vpc-0e9801d129EXAMPLE*

Look up the ARN format for a resource

The exact format of an ARN depends on the service and resource type. Some resource ARNs can include a path, a variable, or a wildcard. To look up the ARN format for a specific AWS resource, open the Service Authorization Reference, open the page for the service, and navigate to the resource types table.

Paths in ARNs

Resource ARNs can include a path. For example, in Amazon S3, the resource identifier is an object name that can include forward slashes (/) to form a path. Similarly, IAM user names and group names can include paths. Only alphanumeric characters and the following characters are allowed in IAM paths: forward slash (/), plus (+), equals (=), comma (,), period (.), at (@), underscore (_), and hyphen (-).

Using wildcards in paths

Paths can include a wildcard character, namely an asterisk (*). Some policy elements allow wildcards while others don't. You can use wildcards for the Resource or NotResource elements but not for the Principal or NotPrincipal elements. For more information, see IAM JSON policy reference.

You can specify `role/*` to mean all roles in the account 123456789012 as in the following example:

```
arn:aws:iam::123456789012:role/*
```

You can also end a resource name with a wildcard. For example, you can specify **`service-*`** to mean all roles starting with `service` and ending with different characters like `service-role1` or `service-test`:

```
arn:aws:iam::123456789012:role/service-*
```

The following example shows ARNs for objects in an Amazon S3 bucket in which the resource name includes a path. The ARN `arn:aws:s3:::amzn-s3-demo-bucket/*` is for all objects within that bucket, regardless of prefix. The ARN `arn:aws:s3:::amzn-s3-demo-bucket/Development/*` is for all objects created within the **`/Development/`** prefix.

You can also use the ? wildcard character to specify one character in an ARN. For example, you could use the following ARN for all folders starting with four characters and ending in **`-test`** in the S3 bucket named amzn-s3-demo-bucket. Some folders that would match this include `1234-test`, `2024-test`, or `a100-test`.

```
arn:aws:s3:::amzn-s3-demo-bucket/????-test
```

You can also use wildcards in the different sections of an ARN, delimited by a colon "**:**". In the following example, two wildcards are used to match all Amazon Q applications and resources within the applications in all regions for account 123456789012:

```
arn:aws:qbusiness:*:123456789012:*
```

Similarly, the following example matches all Amazon VPCs in all regions for account 123456789012:

```
arn:aws:ec2:*:123456789012:vpc/*
```

The following example matches all Amazon EBS volumes in all regions for account 123456789012:

```
arn:aws:ec2:*:123456789012:volume/*
```

Limitations on wildcard usage within ARNs

You cannot use a wildcard in the portion of the ARN that specifics the resource type. The following example ARN with a wildcard within the resource type is not valid:

```
arn:aws:lambda:us-east-2:123456789012:functi*:my-function <== not allowed
```

> **Note**
>
> When you specify an incomplete ARN (one with fewer than the standard six fields) in an identity-based policy, AWS automatically completes the ARN by adding wildcard

characters (*) to all missing fields. For example, specifying `arn:aws:sqs` is equivalent to `arn:aws:sqs:*:*:*`, which grants access to all Amazon Amazon SQS resources across all regions and accounts. However, session policies passed to AWS STS AssumeRole, AssumeRoleWithWebIdentity, and AssumeRoleWithSAML requests, do not support incomplete ARNs. Using an incomplete ARN in a session policy will result in a `MalformedPolicyDocumentException` error.

You also cannot use a wildcard in the prefix ARN, or have a wildcard in the partition section of an ARN.

```
arn:aws:redshift:us-east-1:123456789012:? <== not allowed
```

IAM identifiers

IAM uses a few different identifiers for users, IAM groups, roles, policies, and server certificates. This section describes the identifiers and when you use each.

Topics

- Friendly names and paths
- IAM ARNs
- Unique identifiers
- Understanding unique ID prefixes
- Getting the unique identifier

Friendly names and paths

When you create a user, a role, a user group, or a policy, or when you upload a server certificate, you give it a friendly name. Examples include Bob, TestApp1, Developers, ManageCredentialsPermissions, or ProdServerCert.

If you use the IAM API or AWS Command Line Interface (AWS CLI) to create IAM resources, you can add an optional path. You can use a single path, or nest multiple paths as a folder structure. For example, you could use the nested path `/division_abc/subdivision_xyz/product_1234/engineering/` to match your company organizational structure. You could then create a policy to allow all users in that path to access the policy simulator API. To view this policy, see IAM: Access

the policy simulator API based on user path. For information about how a friendly name can be specified, see the User API documentation. For additional examples of how you might use paths, see IAM ARNs.

When you use CloudFormation to create resources, you can specify a path for users, IAM groups, and roles, and customer managed policies.

If you have a user and user group in the same path, IAM doesn't automatically put the user in that user group. For example, you might create a Developers user group and specify the path as `/division_abc/subdivision_xyz/product_1234/engineering/`. If you create a user named Bob and add the same path to him, this doesn't automatically put Bob in the Developers user group. IAM doesn't enforce any boundaries between users or IAM groups based on their paths. Users with different paths can use the same resources if they've been granted permission to those resources. The number and size of IAM resources in an AWS account are limited. For more information, see IAM and AWS STS quotas.

IAM ARNs

Most resources have a friendly name for example, a user named Bob or a user group named `Developers`. However, the permissions policy language requires you to specify the resource or resources using the following *Amazon Resource Name (ARN)* format.

```
arn:partition:service:region:account:resource
```

Where:

- `partition` identifies the partition for the resource. For standard AWS Regions, the partition is `aws`. If you have resources in other partitions, the partition is `aws-partitionname`. For example, the partition for resources in the China (Beijing) Region is `aws-cn`. You cannot delegate access between accounts in different partitions.
- `service` identifies the AWS product. IAM resources always use `iam`.
- `region` identifies the Region of the resource. For IAM resources, this is always kept blank.
- `account` specifies the AWS account ID with no hyphens.
- `resource` identifies the specific resource by name.

You can specify IAM and AWS STS ARNs using the following syntax. The Region portion of the ARN is blank because IAM resources are global.

Syntax:

```
arn:aws:iam::account:root
arn:aws:iam::account:user/user-name-with-path
arn:aws:iam::account:group/group-name-with-path
arn:aws:iam::account:role/role-name-with-path
arn:aws:iam::account:policy/policy-name-with-path
arn:aws:iam::account:instance-profile/instance-profile-name-with-path
arn:aws:sts::account:federated-user/user-name
arn:aws:sts::account:assumed-role/role-name/role-session-name
arn:aws:sts::account:self
arn:aws:iam::account:mfa/virtual-device-name-with-path
arn:aws:iam::account:u2f/u2f-token-id
arn:aws:iam::account:server-certificate/certificate-name-with-path
arn:aws:iam::account:saml-provider/provider-name
arn:aws:iam::account:oidc-provider/provider-name
arn:aws:iam::aws:contextProvider/context-provider-name
```

Many of the following examples include paths in the resource part of the ARN. Paths cannot be created or manipulated in the AWS Management Console. To use paths, you must work with the resource by using the AWS API, the AWS CLI, or the Tools for Windows PowerShell.

Examples:

```
arn:aws:iam::123456789012:root
arn:aws:iam::123456789012:user/John
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Jane
arn:aws:iam::123456789012:group/Developers
arn:aws:iam::123456789012:group/division_abc/subdivision_xyz/product_A/Developers
arn:aws:iam::123456789012:role/S3Access
arn:aws:iam::123456789012:role/application_abc/component_xyz/RDSAccess
arn:aws:iam::123456789012:role/aws-service-role/access-analyzer.amazonaws.com/
AWSServiceRoleForAccessAnalyzer
arn:aws:iam::123456789012:role/service-role/QuickSightAction
arn:aws:iam::123456789012:policy/UsersManageOwnCredentials
arn:aws:iam::123456789012:policy/division_abc/subdivision_xyz/UsersManageOwnCredentials
arn:aws:iam::123456789012:instance-profile/Webserver
arn:aws:sts::123456789012:federated-user/John
arn:aws:sts::123456789012:assumed-role/Accounting-Role/Jane
arn:aws:sts::123456789012:self
arn:aws:iam::123456789012:mfa/JaneMFA
arn:aws:iam::123456789012:u2f/user/John/default (U2F security key)
```

```
arn:aws:iam::123456789012:server-certificate/ProdServerCert
arn:aws:iam::123456789012:server-certificate/division_abc/subdivision_xyz/
ProdServerCert
arn:aws:iam::123456789012:saml-provider/ADFSProvider
arn:aws:iam::123456789012:oidc-provider/GoogleProvider
arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-west-2.amazonaws.com/id/
a1b2c3d4567890abcdefEXAMPLE11111
arn:aws:iam::123456789012:oidc-provider/server.example.org
arn:aws:iam::aws:contextProvider/IdentityCenter
```

The following examples provide more detail to help you understand the ARN format for different types of IAM and AWS STS resources.

- An IAM user in the account:

> **Note**
>
> Each IAM user name is unique. The user name is case-insensitive for the user, such as during the sign in process, but is case-sensitive when you use it in a policy or as part of an ARN.

```
arn:aws:iam::123456789012:user/John
```

- Another user with a path reflecting an organization chart:

```
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/Jane
```

- An IAM user group:

```
arn:aws:iam::123456789012:group/Developers
```

- An IAM user group with a path:

```
arn:aws:iam::123456789012:group/division_abc/subdivision_xyz/product_A/Developers
```

- An IAM role:

```
arn:aws:iam::123456789012:role/S3Access
```

- A service-linked role:

```
arn:aws:iam::123456789012:role/aws-service-role/access-analyzer.amazonaws.com/
AWSServiceRoleForAccessAnalyzer
```

- A service role:

```
arn:aws:iam::123456789012:role/service-role/QuickSightAction
```

- A managed policy:

```
arn:aws:iam::123456789012:policy/ManageCredentialsPermissions
```

- An instance profile that can be associated with an Amazon EC2 instance:

```
arn:aws:iam::123456789012:instance-profile/Webserver
```

- An AWS STS federated user principal identified in IAM as "Paulo":

```
arn:aws:sts::123456789012:federated-user/Paulo
```

- The active session of someone assuming the role of "Accounting-Role", with a role session name of "Mary":

```
arn:aws:sts::123456789012:assumed-role/Accounting-Role/Mary
```

- Represents the caller's own session when used as a resource in an API call, such as the AWS STS SetContext API, that operates on the calling session:

```
arn:aws:sts::123456789012:self
```

- The multi-factor authentication device assigned to the user named Jorge:

```
arn:aws:iam::123456789012:mfa/Jorge
```

- A server certificate:

```
arn:aws:iam::123456789012:server-certificate/ProdServerCert
```

- A server certificate with a path that reflects an organization chart:

```
arn:aws:iam::123456789012:server-certificate/division_abc/subdivision_xyz/
ProdServerCert
```

- Identity providers (SAML and OIDC):

```
arn:aws:iam::123456789012:saml-provider/ADFSProvider
arn:aws:iam::123456789012:oidc-provider/GoogleProvider
arn:aws:iam::123456789012:oidc-provider/server.example.org
```

- OIDC identity provider with a path that reflects an Amazon EKS OIDC identity provider URL:

```
arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-west-2.amazonaws.com/id/
a1b2c3d4567890abcdefEXAMPLE11111
```

- The context provider ARN used with AWS IAM Identity Center trusted identity propagation, from which the trusted context assertion was generated:

```
arn:aws:iam::aws:contextProvider/IdentityCenter
```

Another important ARN is the root user ARN. Although this is not an IAM resource, you should be familiar with the format of this ARN. It is often used in the `Principal` element of a resource-based policy.

- The AWS account displays the following:

```
arn:aws:iam::123456789012:root
```

The following example shows a policy you could assign to Richard to allow him to manage his access keys. Notice that the resource is the IAM user Richard.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRichardAccessKeys",
```

```
            "Effect": "Allow",
            "Action": [
                "iam:*AccessKey*",
                "iam:GetUser"
            ],
            "Resource": "arn:aws:iam::*:user/division_abc/subdivision_xyz/
Richard"
        },
        {
            "Sid": "ListForConsole",
            "Effect": "Allow",
            "Action": "iam:ListUsers",
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> When you use ARNs to identify resources in an IAM policy, you can include *policy variables*. Policy variables can include placeholders for runtime information (such as the user's name) as part of the ARN. For more information, see IAM policy elements: Variables and tags

Using wildcards and paths in ARNs

You can use wildcards in the *`resource`* portion of the ARN to specify multiple users or IAM groups or policies. For example, to specify all users working on product_1234, you use:

```
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/product_1234/*
```

If you have users whose names start with the string app_, you could refer to them all with the following ARN.

```
arn:aws:iam::123456789012:user/division_abc/subdivision_xyz/product_1234/app_*
```

To specify all users, IAM groups, or policies in your AWS account, use a wildcard after the `user/`, `group/`, or `policy/` part of the ARN, respectively.

```
arn:aws:iam::123456789012:user/*
```

```
arn:aws:iam::123456789012:group/*
arn:aws:iam::123456789012:policy/*
```

If you specify the following ARN for a user `arn:aws:iam::111122223333:user/*` it matches both of the following examples.

```
arn:aws:iam::111122223333:user/John
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/JaneDoe
```

But, if you specify the following ARN for a user `arn:aws:iam::111122223333:user/division_abc*` it matches the second example, but not the first.

```
arn:aws:iam::111122223333:user/John
arn:aws:iam::111122223333:user/division_abc/subdivision_xyz/Jane
```

Don't use a wildcard in the `user/`, `group/`, or `policy/` part of the ARN. For example, IAM does not allow the following:

```
arn:aws:iam::123456789012:u*
```

> **Note**
>
> When you specify an incomplete ARN (one with fewer than the standard six fields) in an identity-based policy, AWS automatically completes the ARN by adding wildcard characters (*) to all missing fields. For example, specifying `arn:aws:sqs` is equivalent to `arn:aws:sqs:*:*:*`, which grants access to all Amazon Amazon SQS resources across all regions and accounts. However, session policies passed to AWS STS AssumeRole, AssumeRoleWithWebIdentity, and AssumeRoleWithSAML requests, do not support incomplete ARNs. Using an incomplete ARN in a session policy will result in a `MalformedPolicyDocumentException` error.

Example use of paths and ARNs for a project-based user group

Paths cannot be created or manipulated in the AWS Management Console. To use paths you must work with the resource by using the AWS API, the AWS CLI, or the Tools for Windows PowerShell.

In this example, Jules in the Marketing_Admin user group creates a project-based user group within the /marketing/ path. Jules assigns users from different parts of the company to the user group. This example illustrates that a user's path isn't related to the user groups the user is in.

The marketing group has a new product they'll be launching, so Jules creates a new user group in the /marketing/ path called Widget_Launch. Jules then assigns the following policy to the user group, which gives the user group access to objects in the part of the `example_bucket` that is designated to this particular launch.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::example_bucket/marketing/newproductlaunch/widget/
*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket*",
      "Resource": "arn:aws:s3:::example_bucket",
      "Condition": {"StringLike": {"s3:prefix": "marketing/newproductlaunch/
widget/*"}}
    }
  ]
}
```

Jules then assigns the users who are working on this launch to the user group. This includes Patricia and Eli from the /marketing/ path. It also includes Chris and Chloe from the /sales/ path, and Alice and Jim from the /legal/ path.

Unique identifiers

When IAM creates a user, user group, role, policy, instance profile, or server certificate, it assigns a unique ID to each resource. The unique ID looks like this:

AIDAJQABLZS4A3QDU576Q

For the most part, you use friendly names and ARNs when you work with IAM resources. That way you don't need to know the unique ID for a specific resource. However, the unique ID can sometimes be useful when it isn't practical to use friendly names.

One example reuses friendly names in your AWS account. Within your account, a friendly name for a user, user group, role, or policy must be unique. For example, you might create an IAM user named `John`. Your company uses Amazon S3 and has a bucket with folders for each employee. IAM user `John` is a member of an IAM user group named `User-S3-Access` with permissions that allows users access only to their own folders in the bucket. For an example of how you might create an identity-based policy that allows IAM users to access their own bucket object in S3 using the friendly name of users, see Amazon S3: Allows IAM users access to their S3 home directory, programmatically and in the console.

Suppose that the employee named John leaves your company and you delete the corresponding IAM user named `John`. But later another employee named John starts, and you create a new IAM user named `John`. You add the new IAM user named `John` to the existing IAM user group `User-S3-Access`. If the policy associated to the user group specifies the friendly IAM user name `John`, the policy allows the new John to access information that was left by the former John.

In general, we recommend that you specify the ARN for the resource in your policies instead of its unique ID. However, every IAM user has a unique ID, even if you create a new IAM user that reuses a friendly name you deleted before. In the example, the old IAM user `John` and the new IAM user `John` have different unique IDs. You can create resource-based policies that grant access by unique ID and not just by user name. Doing so reduces the chance that you could inadvertently grant access to information that an employee should not have.

The following example shows how you might specify unique IDs in the `Principal` element of a resource-based policy.

```
"Principal": {
  "AWS": [
    "arn:aws:iam::111122223333:role/role-name",
    "AIDACKCEVSQ6C2EXAMPLE",
    "AROADBQP57FF2AEXAMPLE"
  }
```

The following example shows how you might specify unique IDs in the `Condition` element of a policy using global condition key `aws:userid`.

```
"Condition": {
```

```
    "StringLike": {
      "aws:userId": [
        "AIDACKCEVSQ6C2EXAMPLE",
        "AROADBQP57FF2AEXAMPLE:role-session-name",
        "AROA1234567890EXAMPLE:*",
        "111122223333"
      ]
    }
  }
```

Another example where user IDs can be useful is if you maintain your own database (or other store) of IAM user or role information. The unique ID can provide a unique identifier for each IAM user or role you create. This is the case when you have IAM users or roles that reuse a name, as in the previous example.

Understanding unique ID prefixes

IAM uses the following prefixes to indicate what type of resource each unique ID applies to. Prefixes may vary based on when they were created.

Prefix	Resource type
ABIA	AWS STS service bearer token
ACCA	Context-specific credential
AGPA	User group
AIDA	IAM user
AIPA	Amazon EC2 instance profile
AKIA	Access key
ANPA	Managed policy
ANVA	Version in a managed policy
APKA	Public key
AROA	Role

Prefix	Resource type
ASCA	Certificate
ASIA	Temporary (AWS STS) access key IDs use this prefix, but are unique only in combination with the secret access key and the session token.

Getting the unique identifier

The unique ID for an IAM resource is not available in the IAM console. To get the unique ID, you can use the following AWS CLI commands or IAM API calls.

AWS CLI:

- get-caller-identity
- get-group
- get-role
- get-user
- get-policy
- get-instance-profile
- get-server-certificate

IAM API:

- GetCallerIdentity
- GetGroup
- GetRole
- GetUser
- GetPolicy
- GetInstanceProfile
- GetServerCertificate

IAM and AWS STS quotas

AWS Identity and Access Management (IAM) and AWS Security Token Service (STS) have quotas that limit the size of objects. This affects how you name an object, the number of objects you can create, and the number of characters you can use when you pass an object.

> **Note**
>
> To get account-level information about IAM usage and quotas, use the GetAccountSummary API operation or the get-account-summary AWS CLI command.

IAM name requirements

IAM names have the following requirements and restrictions:

- Policy documents can contain only the following Unicode characters: horizontal tab (U+0009), linefeed (U+000A), carriage return (U+000D), and characters in the range U+0020 to U+00FF.
- Names of users, groups, roles, policies, instance profiles, server certificates, and paths must be alphanumeric, including the following common characters: plus (+), equals (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). Path names must begin and end with a forward slash (/).
- Names of users, groups, roles, and instance profiles must be unique within the account. They aren't distinguished by case, for example, you can't create groups named both **`ADMINS`** and **`admins`**.
- The external ID value that a third party uses to assume a role must have a minimum of 2 characters and a maximum of 1,224 characters. The value must be alphanumeric without white space. It can also include the following symbols: plus (+), equal (=), comma (,), period (.), at (@), colon (:), forward slash (/), and hyphen (-). For more information about the external ID, see Access to AWS accounts owned by third parties.
- Policy names for inline policies must be unique to the user, group, or role they're embedded in. The names can contain any Basic Latin (ASCII) characters except for the following reserved characters: backward slash (\), forward slash (/), asterisk (*), question mark (?), and white space. These characters are reserved according to RFC 3986, section 2.2.
- User passwords (login profiles) can contain any Basic Latin (ASCII) characters.

- AWS account ID aliases must be unique across AWS products, and must be alphanumeric following DNS naming conventions. An alias must be lowercase, it must not start or end with a hyphen, it can't contain two consecutive hyphens, and it can't be a 12-digit number.

For a list of Basic Latin (ASCII) characters, see the Library of Congress Basic Latin (ASCII) Code Table.

IAM object quotas

Quotas, also referred to as limits in AWS, are the maximum values for the resources, actions, and items in your AWS account. Use Service Quotas to manage your IAM quotas.

For the list of IAM service endpoints and service quotas, see AWS Identity and Access Management endpoints and quotas in the *AWS General Reference*.

To request a quota increase

1. Follow the sign-in procedure appropriate to your user type as described in the topic How to sign in to AWS in the *AWS Sign-In User Guide* to sign in to the AWS Management Console.
2. Open the Service Quotas console.
3. In the navigation pane, choose **AWS services**.
4. On the navigation bar, choose the **US East (N. Virginia)** Region. Then search for **`IAM`**.
5. Choose **AWS Identity and Access Management (IAM)**, choose a quota, and follow the directions to request a quota increase.

For more information, see Requesting a Quota Increase in the *Service Quotas User Guide*.

To see an example of how to request an IAM quota increase using the Service Quotas console, watch the following video.

Request an IAM quota increase using the Service Quotas console.

You can request an increase to default quotas for adjustable IAM quotas. Requests up to the maximum quota are automatically approved and completed within a few minutes.

The following table lists the resources for which quota increases can be automatically approved.

Resource	Default quota	Maximum quota
Customer managed policies per account	1500	10000
Groups per account	300	500
Instance profiles per account	1000	10000
Managed policies per role	20	25
Managed policies per user	10	20
Managed policies per group	10	10
Role trust policy length	2048 characters	8192 characters
Roles per account	1000	10000
Server certificates per account	20	20
OpenId connect providers per account	100	700

IAM Access Analyzer quotas

For the list of IAM Access Analyzer service endpoints and service quotas, see IAM Access Analyzer endpoints and quotas in the *AWS General Reference*.

IAM Roles Anywhere quotas

For the list of IAM Roles Anywhere service endpoints and service quotas, see AWS Identity and Access Management Roles Anywhere endpoints and quotas in the *AWS General Reference*.

STS request quotas

The AWS Security Token Service (AWS STS) enforces the following request quotas.

For AWS STS requests made using AWS credentials, the default request quota is **600 requests per second**, per account, per Region. The following AWS STS operations share this quota:

- AssumeRole
- DecodeAuthorizationMessage
- GetAccessKeyInfo
- GetCallerIdentity
- GetFederationToken
- GetSessionToken

> **Note**
>
> Requests to AWS STS by AWS service principals, such as those used to assume roles for use with an AWS service, do not consume STS request per second quota in your accounts.

For example, if an AWS account makes 100 GetCallerIdentity requests per second and 100 AssumeRole calls per second in the same region, that account is consuming 200 of its available 600 STS requests per second for that region.

For cross-account AssumeRole requests, only the account making the AssumeRole request impacts the STS quota. The target account does not have any of it's quota consumed.

To request an increase to STS request quotas, please open a ticket with AWS support.

> **Note**
>
> With the upcoming changes to the AWS STS global endpoint (`https://sts.amazonaws.com`), requests to the global endpoint will not share a requests per second (RPS) quota with AWS STS Regional endpoints in Regions that are enabled by default. When a request to the AWS STS global endpoint originates from a single Region, it will count against the global endpoint's RPS quota. However, when requests come from multiple Regions, each additional Region will receive its own independent RPS quota. For more information about the AWS STS global endpoint changes, see AWS STS global endpoint changes.

IAM and STS character limits

The following are the maximum character counts and size limits for IAM and AWS STS. You can't request an increase for the following limits.

Description	Limit
Alias for an AWS account ID	3–63 characters
For inline policies	You can add as many inline policies as you want to an IAM user, role, or group. But the total aggregate policy size (the sum size of all inline policies) per entity can't exceed the following limits: • User policy size can't exceed 2,048 characters. • Role policy size can't exceed 10,240 characters. • Group policy size can't exceed 5,120 characters. **Note** IAM doesn't count white space when calculating the size of a policy against these limits.
For customer managed policies	• The size of each managed policy can't exceed 6,144 characters.

Description	Limit
	Note IAM doesn't count white space when calculating the size of a policy against this limit.
Group name	128 characters
Instance profile name	128 characters
Password for a login profile	1–128 characters
Path	512 characters
Policy name	128 characters
Role name	64 characters **Important** If you intend to use a role with the **Switch Role** feature in the AWS Management Console, then the combined `Path` and `RoleName` can't exceed 64 characters.

Description	Limit
Role session duration	12 hours When you assume a role from the AWS CLI or API, you can use the `duration-seconds` CLI parameter or the `DurationSeconds` API parameter to request a longer role session. You can specify a value from 900 seconds (15 minutes) up to the maximum session duration setting for the role, which can range 1–12 hours. If you don't specify a value for the `DurationSeconds` parameter, your security credentials are valid for one hour. IAM users who switch roles in the console are granted the maximum session duration, or the remaining time in the user's session, whichever is less. The maximum session duration setting doesn't limit sessions assumed by AWS services. To learn how to view the maximum value for your role, see Update the maximum session duration for a role.
Role session name	64 characters

Description	Limit
Role session policies	• The size of the passed JSON policy document and all passed managed policy ARN characters combined can't exceed 2,048 characters. • You can pass a maximum of 10 managed policy ARNs when you create a session. • You can pass only one JSON policy document when you programmatically create a temporary session for a role or AWS STS federated user principal. • Additionally, session policies and session tags add to the session token size, which has **a separate limit**. The `SessionTokenUtilization` response element shows what percentage of the size limit the token uses. The `PackedPolicySize` response element is deprecated. Use `SessionTokenUtilization` instead. • We recommend that you pass session policies using the AWS CLI or AWS API. The AWS Management Console might increase the session token size by adding session information.

Description	Limit
Role session tags	• Session tags must meet the tag key limit of 128 characters and the tag value limit of 256 characters. • You can pass up to 50 session tags. • Session policies and session tags add to the session token size, which has a separate limit. You can pass session tags using the AWS CLI or AWS API. The `SessionTokenUtilization` response element shows what percentage of the size limit the token uses. The `PackedPolicySize` response element is deprecated. Use `SessionTokenUtilization` instead.
SAML authentication response base64 encoded	100,000 characters This character limit applies to `assume-role-with-saml` CLI or `AssumeRoleWithSAML` API operation.
Tag key	128 characters This character limit applies to tags on IAM resources and session tags.
Tag value	256 characters This character limit applies to tags on IAM resources and session tags. Tag values can be empty which means tag values can have a length of 0 characters.

Description	Limit
Unique IDs created by IAM	128 characters. For example: • User IDs that begin with AIDA • Group IDs that begin with AGPA • Role IDs that begin with AROA • Managed policy IDs that begin with ANPA • Server certificate IDs that begin with ASCA **Note** This isn't intended to be an exhaustiv e list, nor is it a guarantee that IDs of a certain type begin only with the specified letter combination.
User name	64 characters

Dual-stack endpoint support

Dual-stack endpoints enable clients to communicate with AWS services using either IPv4 or IPv6 addresses. Both IAM and AWS STS provide dual-stack endpoint support. For more information about how to configure your VPC for dual-stack mode, see IPv6 support for your VPC in the *Amazon VPC User Guide*.

If you use IAM policies that include the `aws:sourceIp` or `aws:vpcSourceIp` condition keys, you need to update these policies to support IPv6 clients. For more information about IPv6 addressing for your VPCs, see IP addressing for your VPCs and subnets in the *Amazon VPC User Guide*.

IAM dual-stack endpoint support

IAM provides a dual-stack public endpoint that supports both IPv4 and IPv6 clients. The IAM dual-stack public endpoint can also be accessed privately from your virtual private cloud (VPC) using AWS PrivateLink. For more information about creating private interface VPC endpoints for IAM, see Create a VPC endpoint for IAM.

The IAM dual-stack public endpoint is `https://iam.global.api.aws`.

The IAM public endpoint at `https://iam.amazonaws.com`, unlike the dual-stack public endpoint, supports only IPv4 clients. When accessed privately from your VPC using AWS PrivateLink, the IAM public endpoint can support both IPv4 and IPv6 clients.

AWS STS dual-stack endpoint support

AWS STS provides dual-stack regional endpoints that support both IPv4 and IPv6 clients. The AWS STS dual-stack regional endpoints can also be accessed privately from your virtual private cloud (VPC) using AWS PrivateLink. For more information about creating private interface VPC endpoints for AWS STS, see Create a VPC endpoint for AWS STS.

The following table shows the AWS STS dual-stack regional endpoints by partition:

Partition	Dual-stack endpoint URL
Commercial AWS Regions	**Regular endpoints:** `https://sts.` *`<region>`*`.api.aws` **FIPS endpoints:** `https://sts-fips.` *`<region>`*`.api.aws`
AWS GovCloud (US) Regions	**Regular endpoints:** `https://sts.` *`<region>`*`.api.aws`
China Regions	**Regular endpoints:** `https://sts.` *`<region>`*`.api.amazonwebservices.com.cn`

> **Note**
>
> FIPS endpoints only apply to US and Canada regions.

Limitations

The following limitations apply to AWS STS dual-stack endpoint support:

- IPv6 clients are not supported on the global endpoint `https://sts.amazonaws.com`. You must use dual-stack regional endpoints for IPv6 client support.
- IPv6-only AWS STS VPC endpoints are not supported. VPC endpoints can be configured for IPv4 or dual-stack connectivity.

Interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and AWS Identity and Access Management (IAM) or AWS Security Token Service (AWS STS). You can use this connection to enable IAM or AWS STS to communicate with your resources in your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such as the IP address range, subnets, route tables, and network gateways. To connect your VPC to IAM or AWS STS, you define an *interface VPC endpoint* for each service. The endpoint provides reliable, scalable connectivity to IAM or AWS STS without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see What Is Amazon VPC? in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by AWS PrivateLink an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see AWS PrivateLink for AWS Services.

The following information is for users of Amazon VPC. For more information, see Getting Started with Amazon VPC in the *Amazon VPC User Guide*.

Topics

- VPC endpoint availability
- Create a VPC endpoint for IAM
- Create a VPC endpoint for AWS STS
- Create a VPC endpoint for AWS STS OIDC discovery
- Control access to AWS STS with VPC endpoint policies
- Use VPC condition keys to control federated access

VPC endpoint availability

Important

Interface VPC endpoints for IAM can only be created in the Region where the IAM control plane is located. If your VPC is in a different Region from the IAM control plane Region,

you must use AWS Transit Gateway to allow access to the IAM interface VPC endpoint from another Region. For more information, see Create a VPC endpoint for IAM.

IAM currently supports VPC endpoints in the following Regions:

- US East (N. Virginia)
- China (Beijing)
- AWS GovCloud (US-West)

AWS STS currently supports VPC endpoints in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Hyderabad)
- Asia Pacific (Jakarta)
- Asia Pacific (Melbourne)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Canada West (Calgary)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)

- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Spain)
- Europe (Stockholm)
- Europe (Zurich)
- Israel (Tel Aviv)
- Middle East (Bahrain)
- Middle East (UAE)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Create a VPC endpoint for IAM

To start using IAM with your VPC, create an interface VPC endpoint for IAM. For more information, see Access an AWS service using an interface VPC endpoint in the *Amazon VPC User Guide*.

Interface VPC endpoints for IAM can only be created in the Region where the IAM control plane is located. In commercial AWS Regions, the IAM control plane is located in the US East (N. Virginia) Region (us-east-1). The AWS PrivateLink interface VPC endpoint service name for IAM is `com.amazonaws.iam`. For a list of AWS Regions that support VPC endpoints for IAM, see VPC endpoint availability.

If your VPC is in a different Region from the IAM control plane Region, you must use AWS Transit Gateway to allow access to the IAM interface VPC endpoint from another Region.

To access an IAM interface VPC endpoint from a VPC in a different Region using AWS Transit Gateway

1. Create a transit gateway, or use an existing transit gateway to interconnect your virtual private clouds (VPCs). A transit gateway is required for each Region. For more information, see Create a transit gateway in the *AWS Transit Gateway Guide*.

2. Create transit gateway VPC attachments to connect each VPC to the transit gateway. For more information, see Create a transit gateway attachment to a VPC in the *AWS Transit Gateway Guide*.
3. Create a transit gateway VPC peering attachment to route traffic between peered VPCs. For more information, see Create a peering attachment in the *AWS Transit Gateway Guide*.

> **Note**
>
> VPC peering connections can also route traffic between peered VPCs, but this method does not scale well with a large number of VPCs. Instead of VPC peering, we recommend AWS Transit Gateway peering attachments which improve VPC and on-premises network management through a scalable central hub. For more information about VPC peering connections, see Work with VPC peering connections in the *Amazon VPC Peering Guide*.

Create a VPC endpoint for AWS STS

To start using AWS STS with your VPC, create an interface VPC endpoint for AWS STS. For more information, see Access an AWS service using an interface VPC endpoint in the *Amazon VPC User Guide*.

After you create the VPC endpoint, you must use the matching regional endpoint to send your AWS STS requests. AWS STS recommends that you use both the `setRegion` and `setEndpoint` methods to make calls to a Regional endpoint. You can use the `setRegion` method alone for manually enabled Regions, such as Asia Pacific (Hong Kong). In this case, the calls are directed to the STS Regional endpoint. To learn how to manually enable a Region, see Managing AWS Regions in the *AWS General Reference*. If you use the `setRegion` method alone for Regions enabled by default, the calls are directed to the global endpoint of `https://sts.amazonaws.com` .

When you use regional endpoints, AWS STS calls other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, assume that you have created an interface VPC endpoint for AWS STS and have already requested temporary credentials from AWS STS from resources that are located in your VPC. In that case, these credentials begin flowing through the interface VPC endpoint by default. For more information about making Regional requests using AWS STS, see Manage AWS STS in an AWS Region.

Create a VPC endpoint for AWS STS OIDC discovery

AWS Security Token Service (AWS STS) OIDC discovery is a pair of public endpoints that serve the metadata and verification keys relying parties need to validate JSON Web Tokens (JWTs) issued by outbound identity federation.

When you enable outbound identity federation, AWS STS publishes an account-specific issuer URL. This URL hosts two OpenID Connect (OIDC) discovery endpoints. Relying parties fetch these endpoints to obtain the metadata and public keys they need to verify the JWTs that GetWebIdentityToken issues:

- `{issuer_url}/.well-known/openid-configuration` — The OIDC discovery document
- `{issuer_url}/.well-known/jwks.json` — The JSON Web Key Set (JWKS) that contains the token signature verification keys

By default, reaching these endpoints from your VPC requires an internet gateway, NAT device, VPN connection, or connection. To keep this traffic on the AWS network instead, create an interface VPC endpoint for the STS OIDC discovery endpoint.

> **Note**
>
> The STS OIDC discovery endpoint serves only the two endpoints listed above. It does not serve any AWS STS API operation. To call `GetWebIdentityToken`, `AssumeRole`, or any other AWS STS operation privately, use the `com.amazonaws.`*`region`*`.sts` endpoint service. See Create a VPC endpoint for AWS STS.

Choosing the right endpoint service

Outbound identity federation uses two distinct network paths. Most workloads that both mint and verify tokens need both endpoint services.

Task	Endpoint service	Authentication
Mint a JWT (`sts:GetWebIdentityToken`)	`com.amazonaws.`*`region`*`.sts`	AWS SigV4

Task	Endpoint service	Authentication
Read `/.well-known/openid-configuration`	`com.amazonaws.region.sts-oidc`	None (unauthenticated)
Read `/.well-known/jwks.json`	`com.amazonaws.region.sts-oidc`	None (unauthenticated)

You do not need the STS OIDC discovery endpoint if the only party that verifies your tokens is an external service outside AWS. That service fetches your issuer URL over the public internet, and its traffic does not enter your VPC.

Considerations

Consider the following before you create an interface VPC endpoint for STS OIDC discovery.

- **VPC endpoint policies are not supported.** Both discovery APIs serve publicly available data, and the STS OIDC discovery endpoint does not enforce VPC endpoint policies. Use security groups and subnet placement to control which resources can reach the endpoint.
- **Only two paths are served.** The endpoint accepts HTTPS `GET` requests to `/.well-known/openid-configuration` and `/.well-known/jwks.json` only. All other paths and methods return an error.

Creating an interface VPC endpoint for STS OIDC discovery

You can create a VPC endpoint for the STS OIDC discovery endpoint using either the Amazon Virtual Private Cloud console or the AWS Command Line Interface. For more information, see Access an AWS service using an interface VPC endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for STS OIDC discovery using the following service name:

- `com.amazonaws.region.sts-oidc`

If you enable private DNS for the endpoint, you can send requests to `sts-oidc.region.amazonaws.com` and they will resolve to the endpoint's network interfaces within your VPC.

Sending requests through the endpoint

When a relying party in your VPC needs to verify a JWT, it fetches the OIDC discovery document and JWKS from the issuer URL. The issuer hostname (for example, `uuid.tokens.sts.global.api.aws`) uses a different domain than the endpoint (`sts-oidc.region.amazonaws.com`), so it does not resolve to the VPC endpoint by default.

To route requests for the issuer hostname through the endpoint, create an Amazon Route 53 private hosted zone for your issuer hostname and add an alias record that points to the VPC endpoint. After you create the hosted zone, resources in the VPC resolve the issuer hostname to the endpoint's network interfaces. Standard OIDC libraries can then fetch the discovery documents with no code changes. For more information about creating a private hosted zone, see Working with private hosted zones in the *Amazon Route 53 Developer Guide*.

> **Note**
>
> The endpoint service is regional, but a single endpoint serves discovery requests for any issuer UUID in the partition. Create endpoints in the Regions where your verifiers run; you do not need one per issuer account.

Control access to AWS STS with VPC endpoint policies

When you create an interface VPC endpoint for AWS Security Token Service (AWS STS), you can attach an endpoint policy. The policy controls which principals can use the endpoint and which AWS STS actions they can perform. If you don't attach a policy, the endpoint uses the default policy that allows unrestricted access to all AWS STS actions for all principals.

VPC endpoint policies don't grant permissions on their own. They act as an additional boundary that works alongside other policies. Both the endpoint policy and the caller's applicable policies must allow a request for it to succeed.

For more information about VPC endpoint policies, see Control access to VPC endpoints using endpoint policies in the *Amazon VPC User Guide*.

Topics

- Default VPC endpoint policy
- Important considerations for AWS STS VPC endpoint policies
- Condition keys available for AWS STS VPC endpoint policies

- Example: Allow all AWS STS actions for your organization
- Example: Allow all AWS STS actions for specific accounts
- Example: Restrict to specific AWS STS actions
- Example: Deny non-organization access while allowing federation

Default VPC endpoint policy

If you don't attach a custom policy when you create the endpoint, AWS attaches the following default policy. This policy allows unrestricted access to the endpoint.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

To limit access to the endpoint, attach a custom endpoint policy.

Important considerations for AWS STS VPC endpoint policies

AWS STS handles requests from two fundamentally different types of callers. Your VPC endpoint policy must account for both types to avoid unintentionally blocking legitimate requests.

Authenticated AWS principals

IAM users and IAM roles that sign requests with AWS Signature Version 4 (SigV4). These callers have standard condition keys such as `aws:PrincipalOrgID`, `aws:PrincipalAccount`, and `aws:PrincipalArn` available in the request context.

Federated callers

SAML 2.0 and OpenID Connect (OIDC) principals that call `AssumeRoleWithSAML` or `AssumeRoleWithWebIdentity`. These callers authenticate with SAML assertions or JSON Web Tokens (JWTs), not SigV4 signatures. Because they have no AWS identity at the time

of the request, the request context does not include principal-based condition keys such as `aws:PrincipalOrgID`, `aws:PrincipalAccount`, and `aws:PrincipalArn`.

> **Important**
>
> If your VPC endpoint policy relies solely on `aws:PrincipalOrgID` to allow access, federated `AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity` calls are implicitly denied because the condition key is absent for non-AWS principals.

How AWS STS evaluates VPC endpoint policies for federated callers

When a federated caller invokes `AssumeRoleWithSAML` or `AssumeRoleWithWebIdentity` through a VPC endpoint, the following applies:

- The caller is not an AWS principal. Condition keys such as `aws:PrincipalOrgID`, `aws:PrincipalAccount`, and `aws:PrincipalArn` are not available in the request context.
- Federated callers do not have the condition key `aws:PrincipalIsAWSService` in the request context.
- The role being assumed is an AWS resource. Resource-based condition keys such as `aws:ResourceOrgID` and `aws:ResourceAccount` are available and refer to the target role.
- The role's trust policy remains the primary authorization gate for federated access. The VPC endpoint policy provides an additional network-level boundary.

We recommend using `aws:ResourceOrgID` or `aws:ResourceAccount` when writing VPC endpoint policy statements that must apply to federated callers, because these callers do not have principal-based condition keys available.

Condition keys available for AWS STS VPC endpoint policies

The following table shows commonly used condition keys and their availability in the request context for each caller type when making requests through an AWS STS VPC endpoint. Additional condition keys are available beyond those listed here.

Condition key availability by caller type

Condition key	Authenticated AWS principals	Federated callers	Description
`aws:PrincipalOrgID`	Yes	No	The organization ID of the calling principal
`aws:PrincipalAccount`	Yes	No	The account ID of the calling principal
`aws:PrincipalArn`	Yes	No	The ARN of the calling principal
`aws:PrincipalIsAWSService`	Yes (evaluates to false)	No (key is absent)	Whether the caller is an AWS service principal
`aws:ResourceOrgID`	Yes	Yes	The organization ID of the account that owns the requested resource
`aws:ResourceAccount`	Yes	Yes	The account ID that owns the requested resource

Note

For authenticated AWS principals (IAM users and roles), `aws:PrincipalIsAWSService` is present in the request context and evaluates to false. For federated callers, this key is absent from the request context entirely. A condition that checks `"Bool": {"aws:PrincipalIsAWSService": "false"}` does not match federated callers because the key is not present.

Example: Allow all AWS STS actions for your organization

The following endpoint policy restricts your AWS STS VPC endpoint to your organization while supporting federated access. It allows all AWS STS actions for authenticated principals in your

organization and separately permits federated callers to assume roles within your organization. Federated callers (`AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity`) require a separate statement because they don't have an `aws:PrincipalOrgID` in the request context.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOrganizationPrincipals",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "sts:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-exampleorgid"
                }
            }
        },
        {
            "Sid": "AllowFederatedAssumeRole",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "sts:AssumeRoleWithSAML",
                "sts:AssumeRoleWithWebIdentity"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceOrgID": "o-exampleorgid"
                }
            }
        }
    ]
}
```

The first statement uses `"Principal": {"AWS": "*"}` with `aws:PrincipalOrgID` to allow authenticated AWS principals from your organization. The second statement uses `"Principal": "*"` to match federated callers and restricts the target role to your organization

using `aws:ResourceOrgID`. The role's trust policy remains the primary control for which federated identities can assume which roles.

For more information about implementing network perimeter controls, see Building a data perimeter on AWS and the data perimeter policy examples on the GitHub website.

Example: Allow all AWS STS actions for specific accounts

The following endpoint policy allows all AWS STS actions for principals in specific accounts. Use this policy when your accounts are not part of an organization. Federated callers are allowed in a separate statement because they don't have an `aws:PrincipalAccount` in the request context.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSpecificAccountPrincipals",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "sts:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": [
                        "111122223333",
                        "444455556666"
                    ]
                }
            }
        },
        {
            "Sid": "AllowFederatedAssumeRole",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "sts:AssumeRoleWithSAML",
                "sts:AssumeRoleWithWebIdentity"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:ResourceAccount": [
                        "111122223333",
                        "444455556666"
                    ]
                }
            }
        }
    ]
}
```

The first statement uses `"Principal": {"AWS": "*"}` with `aws:PrincipalAccount` to allow authenticated AWS principals from the specified accounts. The second statement uses `"Principal": "*"` to match federated callers and restricts the target role to the same accounts using `aws:ResourceAccount`. The role's trust policy remains the primary control for which federated identities can assume which roles.

Example: Restrict to specific AWS STS actions

The following endpoint policy allows only `AssumeRole` for authenticated principals and `AssumeRoleWithSAML` for federated callers, both scoped to roles in your organization.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAssumeRoleByOrgPrincipals",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "sts:AssumeRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-exampleorgid"
                }
            }
        },
        {
            "Sid": "AllowSAMLFederation",
            "Effect": "Allow",
            "Principal": "*",
```

```
            "Action": "sts:AssumeRoleWithSAML",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceOrgID": "o-exampleorgid"
                }
            }
        }
    ]
}
```

This policy blocks other AWS STS actions such as `GetCallerIdentity`, `GetSessionToken`, and `AssumeRoleWithWebIdentity` through this endpoint. Adjust the `Action` elements to match your requirements.

Example: Deny non-organization access while allowing federation

The following endpoint policy uses an explicit deny to block authenticated principals outside your organization, while preserving access for federated callers. This approach starts with a broad allow and adds a targeted deny statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAll",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*"
        },
        {
            "Sid": "DenyNonOrgPrincipals",
            "Effect": "Deny",
            "Principal": {
                "AWS": "*"
            },
            "Action": "sts:*",
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:PrincipalOrgID": "o-exampleorgid"
```

```
                }
            }
        }
    ]
}
```

The deny statement uses `"Principal": {"AWS": "*"}`, which scopes it to authenticated AWS principals only. Federated callers (SAML and OIDC) are not AWS principals and are not matched by this `Principal` element, so the deny does not apply to them. This approach avoids the need for complex `Null` or `Bool` conditions to carve out exceptions for federated callers.

> **Note**
>
> The first statement allows all actions for all principals. The deny in the second statement takes precedence for authenticated AWS principals outside your organization. Federated callers are allowed by the first statement and are unaffected by the deny because the `Principal` element in the deny statement does not match them.

Use VPC condition keys to control federated access

Federated users can assume roles using `AssumeRoleWithSAML` or `AssumeRoleWithWebIdentity` through a VPC endpoint. When they do, you can use VPC-specific condition keys in the role's trust policy or in resource control policies (RCPs) to restrict where these requests can originate. This provides a network-level boundary in addition to the identity-level controls in the trust policy.

Topics

- How VPC condition keys work for federated requests
- Example: Restrict OIDC federation to a specific VPC
- Example: Restrict SAML federation to a specific VPC endpoint
- Example: Restrict OIDC federation by private IP range within a VPC
- Example: Restrict SAML federation to multiple VPCs
- Example: Combine VPC and provider-specific conditions
- Example: Use an RCP to restrict federated access to a specific VPC

How VPC condition keys work for federated requests

The condition keys available in the request context depend on the network path of the federated request.

VPC condition key availability by network path

Condition key	Through VPC endpoint	Over public internet	Description
`aws:SourceVpc`	Yes	No	VPC ID that the request traverses
`aws:SourceVpcArn`	Yes	No	ARN of the VPC that the request traverses
`aws:SourceVpce`	Yes	No	VPC endpoint ID that the request traverses
`aws:VpcSourceIp`	Yes	No	Private IP address of the caller within the VPC
`aws:SourceIp`	No	Yes	Public IP address of the caller

If your trust policy uses `aws:SourceVpc`, `aws:SourceVpcArn`, or `aws:SourceVpce` in an `Allow` statement and the request does not come through a VPC endpoint, the condition does not match and AWS implicitly denies the request. This effectively requires federated requests to traverse the VPC endpoint to succeed.

> **Important**
>
> For requests made through a VPC endpoint, `aws:SourceIp` is not populated. Use `aws:VpcSourceIp` for IP-based restrictions instead.

We recommend using `aws:SourceVpcArn` instead of `aws:SourceVpc` in trust policies. VPC IDs are regionally unique but not globally unique, whereas `aws:SourceVpcArn` includes the Region and account, providing globally unique identification of the VPC.

> **Note**
>
> The audience (aud) value in the following OIDC examples uses `sts.amazonaws.com`, which is the default for Amazon EKS. For other OIDC providers, replace this with the audience value configured for your application (for example, a client ID or application URI).

Example: Restrict OIDC federation to a specific VPC

The following trust policy allows an OIDC provider to assume the role only when the request originates from a specific VPC through a VPC endpoint.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOIDCFromVpc",
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/idp.example.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "idp.example.com:aud": "sts.amazonaws.com",
                    "aws:SourceVpcArn": "arn:aws:ec2:us-west-2:111122223333:vpc/
vpc-111bbb22"
                }
            }
        }
    ]
}
```

AWS implicitly denies requests from outside the specified VPC because the `aws:SourceVpcArn` condition does not match. No explicit deny statement is needed.

Example: Restrict SAML federation to a specific VPC endpoint

The following trust policy allows a SAML provider to assume the role only when the request comes through a specific VPC endpoint.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSAMLFromVpce",
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:saml-provider/ExampleProvider"
            },
            "Action": "sts:AssumeRoleWithSAML",
            "Condition": {
                "StringEquals": {
                    "SAML:aud": "https://signin.aws.amazon.com/saml",
                    "aws:SourceVpce": "vpce-0abcdef1234567890"
                }
            }
        }
    ]
}
```

Using `aws:SourceVpce` instead of `aws:SourceVpc` provides more granular control when you have multiple VPC endpoints in the same VPC and want to restrict access to a specific endpoint.

Example: Restrict OIDC federation by private IP range within a VPC

The following trust policy restricts federation to requests originating from a specific subnet within the VPC. Use `aws:VpcSourceIp` for IP-based restrictions on requests made through a VPC endpoint.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOIDCFromSubnet",
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/idp.example.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "idp.example.com:aud": "sts.amazonaws.com",
```

```
                    "aws:SourceVpcArn": "arn:aws:ec2:us-west-2:111122223333:vpc/
vpc-111bbb22"
                },
                "IpAddress": {
                    "aws:VpcSourceIp": "10.0.1.0/24"
                }
            }
        }
    ]
}
```

Important

Do not use `aws:SourceIp` with private IP ranges for requests through a VPC endpoint. The `aws:SourceIp` key is not available for VPC endpoint requests. Use `aws:VpcSourceIp` instead.

Example: Restrict SAML federation to multiple VPCs

The following trust policy allows a SAML provider to assume the role from any of several VPCs. This is useful when your organization has multiple VPCs across different environments or Regions that need federated access to the same role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSAMLFromMultipleVpcs",
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:saml-provider/ExampleProvider"
            },
            "Action": "sts:AssumeRoleWithSAML",
            "Condition": {
                "StringEquals": {
                    "SAML:aud": "https://signin.aws.amazon.com/saml",
                    "aws:SourceVpcArn": [
                        "arn:aws:ec2:us-west-2:111122223333:vpc/vpc-111aaa11",
                        "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-222bbb22"
                    ]
```

```
                }
            }
        }
    ]
}
```

Example: Combine VPC and provider-specific conditions

You can combine VPC condition keys with provider-specific conditions such as audience (aud) and subject (sub) for fine-grained access control. The following trust policy restricts role assumption to a specific federated user and requires the request to originate from a specific VPC.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOIDCUserFromVpc",
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/idp.example.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "idp.example.com:aud": "sts.amazonaws.com",
                    "idp.example.com:sub": "user123",
                    "aws:SourceVpcArn": "arn:aws:ec2:us-west-2:111122223333:vpc/
vpc-111bbb22"
                }
            }
        }
    ]
}
```

This combines identity-level controls (restricting to a specific federated user) with network-level controls (restricting to a specific VPC), providing defense in depth.

Example: Use an RCP to restrict federated access to a specific VPC

The following resource control policy (RCP) denies `AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity` requests unless they originate from a specific VPC. Unlike trust

policies that apply per-role, a resource control policy (RCP) applies across all roles in the target accounts. This provides a centralized network boundary.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyFederatedAccessOutsideVpc",
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "sts:AssumeRoleWithWebIdentity",
                "sts:AssumeRoleWithSAML"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:SourceVpcArn": "arn:aws:ec2:us-west-2:111122223333:vpc/
vpc-111bbb22"
                },
                "Null": {
                    "aws:SourceVpcArn": "false"
                }
            }
        }
    ]
}
```

The `Null` condition ensures the deny only applies when `aws:SourceVpcArn` is present in the request context (that is, the request came through a VPC endpoint but from the wrong VPC). Without it, requests over the public internet (where `aws:SourceVpcArn` is absent) would also be denied by `StringNotEquals`. If you remove the `Null` check, the deny also applies to requests that do not traverse any VPC endpoint, including requests over the public internet. We recommend testing thoroughly before removing the `Null` check in production environments.

AWS services that work with IAM

The AWS services listed below are grouped alphabetically and include information about what IAM features they support:

- **Service** – You can choose the name of a service to view the AWS documentation about IAM authorization and access for that service.
- **Actions** – You can specify individual actions in a policy. If the service does not support this feature, then **All actions** is selected in the visual editor. In a JSON policy document, you must use `*` in the `Action` element. For a list of actions in each service, see Actions, Resources, and Condition Keys for AWS Services.
- **Resource-level permissions** – You can use ARNs to specify individual resources in the policy. If the service does not support this feature, then **All resources** is chosen in the policy visual editor. In a JSON policy document, you must use `*` in the `Resource` element. Some actions, such as `List*` actions, do not support specifying an ARN because they are designed to return multiple resources. If a service supports this feature for some resources but not others, it is indicated by **Partial** in the table. See the documentation for that service for more information.
- **Resource-based policies** – You can attach resource-based policies to a resource within the service. Resource-based policies include a `Principal` element to specify which IAM identities can access that resource. For more information, see Identity-based policies and resource-based policies.
- **ABAC (authorization based on tags)** – To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

  For more information about defining permissions based on attributes such as tags, see Define permissions based on attributes with ABAC authorization. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC).
- **Temporary credentials** – You can use short-term credentials that you obtain when you sign in using IAM Identity Center, account access manager, switch roles in the console, or that you generate using AWS STS in the AWS CLI or AWS API. You can access services with a **No** value only while using your long-term IAM user credentials. This includes a user name and password or your user access keys. For more information, see Temporary security credentials in IAM.
- **Service-linked roles** – A service-linked role is a special type of service role that gives the service permission to access resources in other services on your behalf. Choose the **Yes** or **Partial** link to see the documentation for services that support these roles. This column does not indicate if the service uses standard service roles. For more information, see Service-linked roles.

- **More information** – If a service doesn't fully support a feature, you can review the footnotes for an entry to view the limitations and links to related information.

Services that work with IAM

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Account access manager	Yes	Yes	No	Yes	Yes	Yes
AWS Account Management	Yes	Yes	No	No	Yes	No
AWS Action Recommendations	Yes	No	No	No	Yes	No
AWS Activate Console	Yes	No	No	No	Yes	No
Amazon AI Operations	Yes	Yes	No	Yes	Yes	No
AWS Amplify Admin	Yes	Yes	No	No	Yes	No
AWS Amplify	Yes	Yes	No	Partial	Yes	No
AWS Amplify UI Builder	Yes	Yes	No	Yes	Yes	No
Apache Kafka APIs for Amazon MSK clusters	Yes	Yes	No	No	Yes	No
Amazon API Gateway	Yes	Yes	Yes	No	Yes	Yes
Amazon API Gateway Management	Yes	Yes	No	Yes	Yes	No
Amazon API Gateway Management V2	Yes	Yes	No	Yes	Yes	No
AWS App Studio	Yes	No	No	No	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS App2Container	Yes	No	No	No	Yes	No
AWS AppConfig	Yes	Yes	No	Yes	Yes	No
AWS AppFabric	Yes	Yes	No	Yes	Yes	No
Amazon AppFlow	Yes	Yes	No	Yes	Yes	No
Amazon AppIntegrations	Yes	Yes	No	Yes	Yes	Yes
Application Auto Scaling	Yes	Yes	No	Yes	Yes	Yes
AWS Application Discovery Arsenal	Yes	No	No	No	Yes	No
AWS Application Discovery Service	Yes	No	No	No	Yes	Yes
AWS Transform MGN	Yes	Yes	No	Yes	Yes	Yes
Amazon Application Recovery Controller - Zonal Shift	Yes	Yes	No	No	Yes	No
AWS Application Transformation Service	Yes	No	No	No	Yes	No
AWS App Mesh	Yes	Yes	No	Yes	Yes	Yes
AWS App Mesh Preview	Yes	Yes	No	No	Yes	Yes
AWS App Runner	Yes	Yes	No	Yes	Yes	Yes
Amazon AppStream 2.0	Yes	Yes	No	Yes	Yes	No
AWS AppSync	Yes	Yes	No	Yes	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon ARC Region Switch	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Artifact	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Athena	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Audit Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Aurora DSQL	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Auto Scaling	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS B2B Data Interchange	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Backup	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Backup Gateway	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Backup Search	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Backup storage	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Batch	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Bedrock	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Bedrock Agentcore	✓ Yes	✓ Yes	✗ No	⚠ Partial	✓ Yes	✗ No
AWS Billing and Cost Management	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Billing and Cost Management Dashboards	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Billing and Cost Management Data Exports	Yes	Yes	No	Yes	Yes	No
AWS Billing and Cost Management Pricing Calculator	Yes	Yes	No	Yes	Yes	No
AWS Billing and Cost Management Recommended Actions	Yes	No	No	No	Yes	No
AWS Billing Conductor	Yes	Yes	No	Yes	Yes	No
Amazon Braket	Yes	Yes	No	Yes	Yes	Yes
AWS Budget Service	Yes	Yes	No	Yes	Yes	No
AWS Certificate Manager (ACM)	Yes	Yes	No	Yes	Yes	Yes
Amazon Q Developer in chat applications	Yes	Yes	No	Yes	Yes	Yes
Amazon Chime	Yes	Yes	No	Yes	Yes	Yes
AWS Clean Rooms	Yes	Yes	No	Yes	Yes	No
AWS Clean Rooms ML	Yes	Yes	No	Yes	Yes	No
AWS Client VPN	Yes	Yes	No	No	Yes	Yes
AWS Cloud9	Yes	Yes	Yes	Yes	Yes	Yes
AWS Cloud Control API	Yes	No	No	No	Yes	No
Amazon Cloud Directory	Yes	Yes	No	No	Yes	No
CloudFormation	Yes	Yes	No	Yes	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon CloudFront	Yes	Yes	No	Partial	Yes	Yes
Amazon CloudFront KeyValueStore	Yes	Yes	No	Yes	Yes	No
AWS CloudHSM	Yes	Yes	No	Yes	Yes	Yes
AWS Cloud Map	Yes	Yes	No	Yes	Yes	No
Amazon CloudSearch	Yes	Yes	No	No	Yes	No
AWS CloudShell	Yes	Yes	No	No	Yes	No
AWS CloudTrail	Yes	Yes	Partial (Info)	Yes	Yes	Yes
AWS CloudTrail Data	Yes	Yes	No	Yes	Yes	No
Amazon CloudWatch	Yes	Yes	No	Yes	Yes	Partial (Info)
Amazon CloudWatch Application Insights	Yes	No	No	No	Yes	No
Amazon CloudWatch Application Signals	Yes	Yes	No	Yes	Yes	No
Amazon CloudWatch Internet Monitor	Yes	Yes	No	Yes	Yes	No
Amazon CloudWatch Logs	Yes	Yes	Yes	Partial	Yes	Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon CloudWatch Network Synthetic Monitor	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch Observability Access Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch Observability Admin Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon CloudWatch RUM	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CloudWatch Synthetics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeArtifact	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS CodeBuild	✓ Yes	✓ Yes	✓ Yes (Info)	⚠ Partial (Info)	✓ Yes	✗ No
Amazon CodeCatalyst	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS CodeCommit	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeConnections	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeDeploy	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS CodeDeploy secure host commands service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon CodeGuru Profiler	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CodeGuru Reviewer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon CodeGuru Security	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS CodePipeline	Yes	Partial	No	Yes	Yes	No
AWS CodeStar	Yes	Partial	No	Yes	Yes	No
AWS CodeStar Connections	Yes	Yes	No	Yes	Yes	Yes
AWS CodeStar Notifications	Yes	Yes	No	Yes	Yes	Yes
Amazon CodeWhisperer	Yes	Yes	No	Yes	Yes	Yes
Amazon Cognito	Yes	Yes	No	Yes	Yes	Yes
Amazon Cognito Sync	Yes	Yes	No	No	Yes	Yes
Amazon Cognito user pools	Yes	Yes	No	Yes	Yes	Yes
Amazon Comprehend	Yes	Yes	No	Yes	Yes	No
Amazon Comprehend Medical	Yes	No	No	No	Yes	No
AWS Compute Optimizer	Yes	No	No	No	Yes	Yes
AWS Config	Yes	Partial (Info)	No	Yes	Yes	Yes
Amazon Connect Customer	Yes	Yes	No	Yes	Yes	Yes
Amazon Connect Customer Cases	Yes	Yes	No	Yes	Yes	No
Amazon Connect Customer Customer Profiles	Yes	Yes	No	Yes	Yes	Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Connect Customer Outbound Campaigns	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Connect Customer Voice ID	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Connect Health	✓ Yes	✓ Yes	✗ No	⚠ Partial	✓ Yes	✗ No
AWS Console Mobile Application	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Consolidated Billing	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Control Catalog	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Control Tower	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Cost and Usage Report	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Cost Explorer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Cost Optimization Hub	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Customer Verification Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Database Migration Service	✓ Yes	✓ Yes	✗ No (Info)	✓ Yes	✓ Yes	✓ Yes
Database Query Metadata Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Data Exchange	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Data Lifecycle Manager	Yes	Yes	No	Yes	Yes	No
AWS Data Pipeline	Yes	Yes	No	Partial	Yes	No
AWS DataSync	Yes	Yes	No	Yes	Yes	Yes
Amazon DataZone	Yes	Yes	No	Yes	Yes	No
AWS Deadline Cloud	Yes	Yes	No	Yes	Yes	No
AWS DeepRacer	Yes	Yes	No	Yes	Yes	Yes
Amazon Detective	Yes	Yes	No	Yes	Yes	No
AWS Device Farm	Yes	Yes	No	Yes	Yes	Yes
Amazon DevOps Guru	Yes	Yes	No	No	Yes	Yes
AWS Diagnostic tools	Yes	Yes	No	Yes	Yes	No
AWS Direct Connect	Yes	Yes	No	Yes	Yes	Yes
AWS Directory Service	Yes	Yes	No	Yes	Yes	Yes
AWS Directory Service Data	Yes	Yes	No	Yes	Yes	No
Amazon DocumentDB Elastic Clusters	Yes	Yes	No	Yes	Yes	Yes
Amazon DynamoDB Accelerator (DAX)	Yes	Yes	No	No	Yes	Yes
Amazon DynamoDB	Yes	Yes	Yes	Yes	Yes	Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Elastic Compute Cloud (Amazon EC2)	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	⚠ Partial (Info)
Amazon EC2 Auto Scaling	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
EC2 Image Builder	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon EC2 Instance Connect	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Elastic Beanstalk	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic Block Store (Amazon EBS)	✓ Yes	⚠ Partial	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Elastic Container Registry (Amazon ECR)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic Container Registry Public (Amazon ECR Public)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Elastic Container Service (Amazon ECS)	✓ Yes	⚠ Partial (Info)	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Elastic Disaster Recovery	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Elastic File System (Amazon EFS)	✓ Yes	✓ Yes	✓ Yes	⚠ Partial	✓ Yes	✓ Yes
Amazon Elastic Kubernetes Service (Amazon EKS)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Elastic Kubernetes Service (Amazon EKS) Auth	Yes	Yes	No	No	Yes	No
AWS Elastic Load Balancing	Yes	Partial	No	Partial	Yes	Yes
Amazon Elastic VMware Service	Yes	Yes	No	Yes	Yes	Yes
Amazon ElastiCache	Yes	Yes	No	Yes	Yes	Yes
AWS Elemental Appliances and Software Activation Service	Yes	Yes	No	Yes	Yes	No
AWS Elemental Appliances and Software	Yes	Yes	No	Yes	Yes	No
AWS Elemental MediaConnect	Yes	Yes	No	No	Yes	Yes
AWS Elemental MediaConvert	Yes	Yes	No	Yes	Yes	No
AWS Elemental MediaLive	Yes	Yes	No	Yes	Yes	No
AWS Elemental MediaPackage	Yes	Yes	No	Yes	Yes	Partial (Info)
AWS Elemental MediaPackage V2	Yes	Yes	No	Yes	Yes	No
AWS Elemental MediaPackage VOD	Yes	Yes	No	Yes	Yes	Partial (Info)
AWS Elemental MediaTailor	Yes	Yes	No	Yes	Yes	Yes
AWS Elemental Support Cases	Yes	Yes	No	Yes	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Elemental Support Content	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon EMR	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon EMR on EKS	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon EMR Serverless	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS End User Messaging SMS and Voice V2	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS End User Messaging Social	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Entity Resolution	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon EventBridge	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon EventBridge Pipes	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon EventBridge Scheduler	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon EventBridge Schemas	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Fault Injection Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon FinSpace	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon FinSpace API	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Firewall Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial
Fleet Hub for AWS IoT Device Management	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Forecast	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Fraud Detector	Yes	Yes	No	Yes	Yes	No
FreeRTOS	Yes	Yes	No	Yes	Yes	No
AWS Free Tier	Yes	No	No	No	Yes	No
Amazon FSx	Yes	Yes	No	Yes	Yes	Yes
Amazon GameLift Servers	Yes	Yes	No	Yes	Yes	No
Amazon GameLift Streams	Yes	Yes	No	Yes	Yes	No
AWS Global Accelerator	Yes	Yes	No	Yes	Yes	Yes
AWS Glue	Yes	Yes	Partial	Partial	Yes	No
AWS Glue DataBrew	Yes	Yes	No	Yes	Yes	No
AWS Ground Station	Yes	Yes	No	Yes	Yes	Yes
Amazon Ground Truth Labeling	Yes	No	No	No	Yes	No
Amazon GuardDuty	Yes	Yes	No	Yes	Yes	Yes
AWS Health APIs And Notifications	Yes	Yes	No	No	Yes	No
AWS HealthImaging	Yes	Yes	No	Yes	Yes	No
AWS HealthLake	Yes	Yes	No	Yes	Yes	No
AWS HealthOmics	Yes	Yes	No	Yes	Yes	No
AWS IAM Identity Center	Yes	Yes	No	Partial	Yes	Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
IAM Identity Center Directory	Yes	No	No	No	Yes	No
IAM Identity Center Identity Store	Yes	Yes	No	No	Yes	No
IAM Identity Center OIDC service	Yes	Yes	No	No	Yes	No
AWS Identity and Access Management (IAM)	Yes	Yes	Partial (Info)	Partial (Info)	Partial (Info)	No
AWS Identity and Access Management and Access Analyzer	Yes	Yes	No	Yes	Yes	Partial
AWS Identity and Access Management Roles Anywhere	Yes	Yes	No	Yes	Yes	Yes
AWS Identity Store Auth	Yes	No	No	No	Yes	No
AWS Identity Sync	Yes	Yes	No	No	Yes	No
AWS Import/Export	Yes	No	No	No	Yes	No
Amazon Inspector	Yes	Yes	No	Yes	Yes	Yes
Amazon Inspector Classic	Yes	No	No	No	Yes	Yes
Amazon InspectorScan	Yes	No	No	No	Yes	No
Amazon Interactive Video Service	Yes	Yes	No	Yes	Yes	Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Interactive Video Service Chat	Yes	Yes	No	Yes	Yes	No
AWS Invoicing	Yes	Yes	No	Yes	Yes	No
AWS IoT Analytics	Yes	Yes	No	Yes	Yes	No
AWS IoT	Yes	Yes	Partial (Info)	Yes	Yes	No
AWS IoT Core Device Advisor	Yes	Yes	No	Yes	Yes	No
AWS IoT Device Tester	Yes	No	No	No	Yes	No
AWS IoT Events	Yes	Yes	No	Yes	Yes	No
AWS IoT FleetWise	Yes	Yes	No	Yes	Yes	No
AWS IoT Greengrass	Yes	Yes	No	Yes	Yes	No
AWS IoT Greengrass V2	Yes	Yes	No	Partial	Yes	No
AWS IoT Jobs DataPlane	Yes	Yes	No	No	Yes	No
AWS IoT Managed Integrations Service	Yes	Yes	No	Yes	Yes	Yes
AWS IoT SiteWise	Yes	Yes	No	Yes	Yes	Yes
AWS IoT TwinMaker	Yes	Yes	No	Yes	Yes	Yes
AWS IoT Wireless	Yes	Yes	No	Yes	Yes	No
Amazon Kendra	Yes	Yes	No	Yes	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Kendra Intelligent Ranking	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Key Management Service (AWS KMS)	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Amazon Keyspaces (for Apache Cassandra)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Managed Service for Apache Flink	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Managed Service for Apache Flink V2	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Data Firehose	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Kinesis Data Streams	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon Kinesis Video Streams	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Lake Formation	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Lambda	✓ Yes	✓ Yes	✓ Yes	⚠ Partial (Info)	✓ Yes	⚠ Partial (Info)
AWS Launch Wizard	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Lex	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Lex V2	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes
AWS License Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS License Manager Linux Subscriptions Manager	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS License Manager User Subscriptions	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Lightsail	✓ Yes	⚠ Partial (Info)	✗ No	⚠ Partial (Info)	✓ Yes	✓ Yes
Amazon Location Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Location Service Maps	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Location Service Places	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Location Service Routes	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Lookout for Equipment	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Lookout for Metrics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Machine Learning	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Macie	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Mainframe Modernization	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Mainframe Modernization Application Testing	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Managed Blockchain	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Managed Blockchain Query	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Managed Grafana	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Managed Service for Prometheus	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Managed Streaming for Apache Kafka (MSK)	✓ Yes	✓ Yes	⚠ Partial (Info)	✓ Yes	✓ Yes	✓ Yes
Amazon Managed Streaming for Kafka Connect	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Managed Workflows for Apache Airflow	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Marketplace	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Marketplace Catalog	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Marketplace Commerce Analytics	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No
AWS Marketplace Deployment Service	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Marketplace Discovery	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Entitlement Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Marketplace Image Building Service	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Marketplace Management Portal	Yes	No	No	No	Yes	No
AWS Marketplace Metering Service	Yes	No	No	No	Yes	No
AWS Marketplace Private Marketplace	Yes	No	No	No	Yes	No
AWS Marketplace Procurement Systems Integration	Yes	No	No	No	Yes	No
AWS Marketplace Reporting	Yes	Yes	No	No	Yes	No
AWS Marketplace Seller Reporting	Yes	Yes	No	No	Yes	No
AWS Marketplace Vendor Insights	Yes	Yes	No	Yes	Yes	No
Amazon MediaImport	Yes	No	No	No	No	No
Amazon MemoryDB	Yes	Yes	No	Yes	Yes	Yes
Amazon Message Delivery Service	Yes	No	No	No	Yes	No
Amazon Message Gateway Service	Yes	No	No	No	Yes	No
AWS Microservice Extractor for .NET	Yes	No	No	No	Yes	No
AWS Migration Acceleration Program Credits	Yes	Yes	No	No	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Migration Hub	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Migration Hub Orchestrator	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Migration Hub Refactor Spaces	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
AWS Migration Hub Strategy Recommendations	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
Amazon Monitron	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon MQ	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Neptune	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
Multi-party approval	✓ Yes	✓ Yes	✗ No	⚠ Partial (Info)	✓ Yes	✗ No
Amazon Neptune Analytics	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Network Firewall	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Network Flow Monitor	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Network Manager	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes (Info)
AWS Network Manager Chat	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Nimble Studio	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon One Enterprise	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon OpenSearch	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon OpenSearch Ingestion	Yes	Yes	No	Yes	Yes	Yes
Amazon OpenSearch Serverless	Yes	Yes	No	Yes	Yes	Yes
Amazon OpenSearch Service	Yes	Yes	Yes	Yes	Yes	Yes
Oracle Database@AWS	Yes	Yes	No	Yes	Yes	No
AWS OpsWorks	Yes	Yes	No	No	Yes	No
AWS OpsWorks Configuration Management	Yes	Yes	No	No	Yes	No
AWS Organizations	Yes	Yes	Yes	Yes	No	Yes
AWS Outposts	Yes	Yes	No	Yes	Yes	Yes
AWS Parallel Computing Service	Yes	Yes	No	Yes	Yes	Yes
AWS Partner Central account management	Yes	No	No	No	Yes	No
AWS Partner Central Selling	Yes	Yes	No	Yes	Yes	No
AWS Payment Cryptography	Yes	Yes	No	Yes	Yes	No
AWS Payments	Yes	No	No	No	Yes	No
AWS Performance Insights	Yes	Yes	No	Yes	Yes	No
Amazon Personalize	Yes	Yes	No	No	Yes	No
Amazon Pinpoint	Yes	Yes	No	Yes	Yes	No
Amazon Pinpoint Email Service	Yes	Yes	No	Yes	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Pinpoint SMS and Voice Service	Yes	No	No	No	Yes	No
Amazon Polly	Yes	Yes	No	No	Yes	No
AWS Price List	Yes	No	No	No	Yes	No
AWS Private 5G	Yes	Yes	No	Yes	Yes	No
AWS Private CA Connector for Active Directory	Yes	Yes	No	Yes	Yes	No
AWS Private CA Connector for SCEP	Yes	Yes	No	Yes	Yes	No
AWS Private Certificate Authority (AWS Private CA)	Yes	Yes	Yes	Yes	Yes	No
AWS PrivateLink	Yes	No	No	No	Yes	No
AWS Proton	Yes	Yes	No	Yes	Yes	Yes
AWS Purchase Orders Console	Yes	Yes	No	Yes	Yes	No
Amazon Q Business	Yes	Yes	No	Yes	Yes	Yes
Amazon Q Business Q Apps	Yes	Yes	No	Yes	Yes	Yes
Amazon Q Developer	Yes	Yes	No	Yes	Yes	Yes
Amazon Q in Connect	Yes	Yes	No	Yes	Yes	No
Amazon Quantum Ledger Database (Amazon QLDB)	Yes	Yes	No	Yes	Yes	No
AWS Quick	Yes	Yes	No	Yes	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon RDS Data API	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon RDS IAM Authentication	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Recycle Bin	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Redshift	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Redshift Data API	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Redshift Serverless	✓ Yes	✓ Yes	✓ Yes (Info)	✓ Yes	✓ Yes	✗ No
Amazon Rekognition	✓ Yes	✓ Yes	⚠ Partial (Info)	✓ Yes	✓ Yes	✗ No
Amazon Relational Database Service (Amazon RDS) (Info)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS re:Post Private	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Resilience Hub	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Resource Access Manager (AWS RAM)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Resource Explorer	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Resource Groups	✓ Yes	✓ Yes	✗ No	✓ Yes	⚠ Partial (Info)	✓ Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Resource Groups Tagging API	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon RHEL Knowledgebase Portal	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS RoboMaker	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Route 53	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Route 53 Domains	✓ Yes	✗ No	✗ No	✗ No	✗ No	✗ No
Amazon Route 53 Profiles	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Route 53 Recovery Cluster	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Route 53 Recovery Control Config	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon Route 53 Recovery Readiness	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Route 53 Resolver	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS RTB Fabric	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon S3 Express	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
Amazon Glacier	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	⚠ Partial
Amazon S3 Tables	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
Amazon S3 Vectors	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon SageMaker AI	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	⚠ Partial (Info)
Amazon SageMaker AI data science assistant	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon SageMaker AI geospatial capabilities	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon SageMaker AI with MLflow	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS Savings Plans	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Secrets Manager	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✗ No
AWS Security Hub CSPM	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Security Incident Response	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon Security Lake	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Security Token Service (AWS STS)	✓ Yes	⚠ Partial (Info)	✗ No	✓ Yes	⚠ Partial (Info)	✗ No
AWS Serverless Application Repository	✓ Yes	✓ Yes	✓ Yes	✗ No	✓ Yes	✗ No
AWS Service Catalog	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Service for managing AWS Console user experience capabilities	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Service Quotas	Yes	Yes	No	Yes	Yes	No
AWS Shield	Yes	Yes	No	Yes	Yes	Yes
AWS Shield network security director	Yes	No	No	No	Yes	No
AWS Signer	Yes	Yes	No	Yes	Yes	No
AWS Sign-In	Yes	No	Yes	No	Yes	No
Amazon SimpleDB	Yes	Yes	No	No	Yes	No
Amazon Simple Email Service - Mail Manager	Yes	Yes	No	Yes	Yes	Yes
Amazon Simple Email Service (Amazon SES) v2	Yes	Partial (Info)	Yes	Yes	Partial (Info)	Yes
Amazon Simple Notification Service (Amazon SNS)	Yes	Yes	Yes	Yes	Yes	No
Amazon Simple Queue Service (Amazon SQS)	Yes	Yes	Yes	Partial	Yes	No
Amazon Simple Storage Service (Amazon S3)	Yes	Yes	Yes	Partial (Info)	Yes	Partial (Info)
Amazon Simple Storage Service (Amazon S3) Object Lambda	Yes	Yes	No	No	Yes	No
Amazon Simple Storage Service (Amazon S3) on AWS Outposts	Yes	Yes	Yes	No	Yes	Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon Simple Workflow Service (Amazon SWF)	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS SimSpace Weaver	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Site-to-Site VPN	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✓ Yes
AWS Snowball Edge	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Snowball Edge Edge	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Snowball Edge Device Management	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS SQL Workbench	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Step Functions	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Storage Gateway	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Supply Chain	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Support	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✓ Yes
AWS Support App in Slack	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Support Console	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Support Plans	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Sustainability	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Systems Manager	✓ Yes	✓ Yes	⚠ Partial	✓ Yes	✓ Yes	✓ Yes
AWS Systems Manager for SAP	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Systems Manager GUI Connect	Yes	No	No	No	Yes	No
AWS Systems Manager Incident Manager	Yes	Yes	Yes	Yes	Yes	Yes
AWS Systems Manager Incident Manager Contacts	Yes	Yes	Yes	No	Yes	No
AWS Systems Manager Quick Setup	Yes	Yes	No	Yes	Yes	No
Tag Editor	Yes	No	No	No	Yes	No
AWS Tax Settings	Yes	No	No	No	Yes	No
AWS Telco Network Builder	Yes	Yes	No	Yes	Yes	No
Amazon Textract	Yes	No	No	No	Yes	No
Amazon Timestream	Yes	Yes	No	Yes	Yes	No
Amazon Timestream Influxdb	Yes	Yes	No	Yes	Yes	Yes
AWS Tiros API (for Reachability Analyzer)	Yes	No	No	No	No	No
Amazon Transcribe	Yes	Yes	No	Yes	Yes	No
AWS Transfer Family	Yes	Yes	No	Yes	Yes	No
AWS Transform	Yes	Yes	No	No	Yes	No
Amazon Translate	Yes	Yes	No	Yes	Yes	No

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
AWS Trusted Advisor	⚠ Partial (Info)	✓ Yes	✗ No	✗ No	⚠ Partial	✓ Yes
AWS User Notifications	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS User Notifications Contacts	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS User Subscriptions	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
AWS Verified Access	✓ Yes	✗ No	✗ No	✗ No	✓ Yes	✗ No
Amazon Verified Permissions	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
Amazon Virtual Private Cloud (Amazon VPC)	✓ Yes	⚠ Partial (Info)	⚠ Partial (Info)	✓ Yes	✓ Yes	⚠ Partial (Info)
Amazon VPC Lattice	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
Amazon VPC Lattice Services	✓ Yes	✓ Yes	✗ No	✗ No	✓ Yes	✗ No
AWS WAF	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS WAF Classic	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS WAF Regional	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes
AWS Well-Architected Tool	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
AWS Wickr	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✗ No
Amazon WorkMail	✓ Yes	✓ Yes	✗ No	✓ Yes	✓ Yes	✓ Yes

Service	Actions	Resource-level permissions	Resource-based policies	ABAC	Temporary credentials	Service-linked roles
Amazon WorkMail Message Flow	Yes	Yes	No	No	Yes	No
Amazon WorkSpaces	Yes	Yes	No	Yes	Yes	No
AWS WorkSpaces Managed Instances	Yes	Yes	No	Yes	Yes	No
Amazon WorkSpaces Secure Browser	Yes	Yes	No	Yes	Yes	Yes
Amazon WorkSpaces Thin Client	Yes	Yes	No	Yes	Yes	No
AWS X-Ray	Yes	Partial (Info)	No	Partial (Info)	Yes	No

More information

AWS CloudTrail

CloudTrail supports resource-based policies on CloudTrail Lake event data stores, dashboards, and channels used for integrations with event sources outside of AWS.

Amazon CloudWatch

CloudWatch service-linked roles cannot be created using the AWS Management Console, and support only the Alarm Actions feature.

AWS CodeBuild

CodeBuild supports cross-account resource sharing using AWS RAM.

CodeBuild supports ABAC for project-based actions.

AWS Config

AWS Config supports resource-level permissions for multi-account multi-Region data aggregation and AWS Config Rules. For a list of supported resources, see the **Multi-Account Multi-Region Data Aggregation** section and **AWS Config Rules** section of the AWS Config API Guide.

AWS Database Migration Service

You can create and modify policies that are attached to AWS KMS encryption keys you create to encrypt data migrated to supported target endpoints. The supported target endpoints include Amazon Redshift and Amazon S3. For more information, see Creating and Using AWS KMS Keys to Encrypt Amazon Redshift Target Data and Creating AWS KMS Keys to Encrypt Amazon S3 Target Objects in the *AWS Database Migration Service User Guide*.

Amazon Elastic Compute Cloud

Amazon EC2 service-linked roles can be used only for the following features: Spot Instance Requests, Spot Fleet Requests, Amazon EC2 Fleets, and Fast launching for Windows instances.

Amazon Elastic Container Service

Only some Amazon ECS actions support resource-level permissions.

AWS Elemental MediaPackage

MediaPackage supports service-linked roles for publishing customer access logs to CloudWatch but not for other API actions.

AWS Identity and Access Management

IAM supports only one type of resource-based policy called a role *trust policy*, which is attached to an IAM role. For more information, see Grant a user permissions to switch roles.

IAM supports tag-based access control for most IAM resources. For more information, see Tags for AWS Identity and Access Management resources.

Only some of the API actions for IAM can be called with temporary credentials. For more information, see Comparing your API options.

AWS IoT

Devices connected to AWS IoT are authenticated by using X.509 certificates or using Amazon Cognito Identities. You can attach AWS IoT policies to an X.509 certificate or Amazon Cognito Identity to control what the device is authorized to do. For more information, see Security and Identity for AWS IoT in the *AWS IoT Developer Guide*.

AWS Lambda

Lambda supports attribute-based access control (ABAC) for functions, event source mappings, and code signing configurations. Layers are not supported. For more information, see Using attribute-based access control in Lambda.

Lambda doesn't have service-linked roles, but Lambda@Edge does. For more information, see Service-Linked Roles for Lambda@Edge in the *Amazon CloudFront Developer Guide*.

Amazon Lightsail

Lightsail partially supports resource-level permissions and ABAC. For more information, see Actions, resources, and condition keys for Amazon Lightsail.

Amazon Managed Streaming for Apache Kafka (MSK)

You can attach a cluster policy to an Amazon MSK cluster that has been configured for multi-VPC connectivity.

Multi-party approval

For Multi-party approval identity sources and approval teams, you can use all three ABAC condition keys: `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, and `aws:TagKeys`.

For sessions, you can only use the `aws:ResourceTag/key-name` condition key because:

- Sessions are not directly taggable resources.
- Sessions inherit tags from the approval team associated with them.

AWS Network Manager

AWS Cloud WAN also supports service-linked roles. For more information, see AWS Cloud WAN service-linked roles in the *Amazon VPC AWS Cloud WAN Guide*.

Amazon Redshift Serverless

Redshift Serverless supports resource-based policies to share snapshots.

Amazon Relational Database Service

Amazon Aurora is a fully managed relational database engine that's compatible with MySQL and PostgreSQL. You can choose the Aurora MySQL or Aurora PostgreSQL as the DB engine option when setting up new database servers through Amazon RDS. For more information, see Identity and access management for Amazon Aurora in the *Amazon Aurora User Guide*.

Amazon Rekognition

Resource-based policies are only supported for copying Amazon Rekognition Custom Labels models.

AWS Resource Groups

Users can assume a role with a policy that allows Resource Groups operations.

Amazon SageMaker AI

Service-linked roles are currently available for SageMaker AI Studio and SageMaker AI training jobs.

AWS Security Token Service

AWS STS does not have "resources," but does allow restricting access in a similar way to users. For more information, see Denying Access to Temporary Security Credentials by Name.

Only some of the API operations for AWS STS support calling with temporary credentials. For more information, see Comparing your API options.

Amazon Simple Email Service

You can only use resource-level permissions in policy statements that refer to actions related to sending email, such as `ses:SendEmail` or `ses:SendRawEmail`. For policy statements that refer to any other actions, the Resource element can only contain `*`.

Only the Amazon SES API supports temporary security credentials. The Amazon SES SMTP interface does not support SMTP credentials that are derived from temporary security credentials.

Amazon Simple Storage Service

Amazon S3 supports tag-based authorization for directory buckets, object resources, Amazon S3 Storage Lens, Storage Lens groups, and access points.

Amazon S3 supports service-linked roles for Amazon S3 Storage Lens.

AWS Trusted Advisor

API access to Trusted Advisor is through the Support API and is controlled by Support IAM policies.

Amazon Virtual Private Cloud

Amazon VPC supports attaching a single resource policy to a VPC endpoint to restrict what can be accessed through that endpoint. For more information about using resource-based policies to control access to resources from specific Amazon VPC endpoints, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

AWS X-Ray

X-Ray does not support resource-level permissions for all actions.

X-Ray supports tag-based access control for groups and sampling rules.

AWS Signature Version 4 for API requests

Important

If you use an AWS SDK (see Sample Code and Libraries) or AWS Command Line Interface (AWS CLI) tool to send API requests to AWS, you can skip the signature process, as the SDK and CLI clients authenticate your requests by using the access keys that you provide. Unless you have a good reason not to, we recommend that you always use an SDK or the CLI.
In Regions that support multiple signature versions, manually signing requests means you must specify which signature version to use. When you supply requests to Multi-Region Access Points, SDKs and the CLI automatically switch to using Signature Version 4A without additional configuration.

Authentication information that you send in a request must include a signature. AWS Signature Version 4 (SigV4) is the AWS signing protocol for adding authentication information to AWS API requests.

You don't use your secret access key to sign API requests. Instead, you use the SigV4 signing process. Signing requests involves:

1. Creating a canonical request based on the request details.
2. Calculating a signature using your AWS credentials.
3. Adding this signature to the request as an Authorization header.

AWS then replicates this process and verifies the signature, granting or denying access accordingly.

Symmetric SigV4 requires you to derive a key that is scoped to a single AWS service, in a single AWS region, on a particular day. This makes the key and calculated signature different for each region, meaning you must know the region the signature is destined for.

Asymmetric Signature Version 4 (SigV4a) is an extension that supports signing with a new algorithm, and generating individual signatures that are verifiable in more than one AWS region. With SigV4a, you can sign a request for multiple regions, with seamless routing and failover between regions. When you use the AWS SDK or AWS CLI to invoke functionality that requires multi-region signing, the signature type is automatically changed to use SigV4a. For details, see How AWS SigV4a works.

How AWS SigV4 works

The following steps describe the general process of computing a signature with SigV4:

1. The **string to sign** depends on the request type. For example, when you use the HTTP Authorization header or the query parameters for authentication, you use a combination of request elements to create the string to sign. For an HTTP POST request, the POST policy in the request is the string you sign.
2. The **signing key** is a series of calculations, with the result of each step fed into the next. The final step is the signing key.
3. When an AWS service receives an authenticated request, it recreates the **signature** using the authentication information contained in the request. If the signatures match, the service processes the request. Otherwise, it rejects the request.

For more information, see Elements of an AWS API request signature.

How AWS SigV4a works

SigV4a uses asymmetric signatures based on public-private key cryptography. SigV4a goes through a similar scoped credentials derivation process as SigV4, except SigV4a uses the same key to sign all requests without needing to derive a distinct signing key based on the date, service, and region. An Elliptic Curve Digital Signature Algorithm (ECDSA) keypair can be derived from your existing AWS secret access key.

The system uses asymmetric cryptography to verify multi-region signatures, so that AWS only needs to store your public keys. Public keys are not secret and can't be used to sign requests. Asymmetric signatures are required for multi-region API requests, such as with Amazon S3 Multi-Region Access Points.

The following steps describe the general process of computing a signature with SigV4a:

1. The **string to sign** depends on the request type. For example, when you use the HTTP Authorization header or the query parameters for authentication, you use a combination of request elements to create the string to sign. For an HTTP POST request, the P0ST policy in the request is the string you sign.
2. The **signing key** is derived from an AWS secret access key through a series of calculations, with the result of each step fed into the next. The final step produces the keypair.
3. When an AWS service receives a request signed with SigV4a, AWS verifies the signature using only the public half of the keypair. If the signature is valid, the request is authenticated and the service processes the request. Requests with invalid signatures are rejected.

For more information about SigV4a for multi-Region API requests, see the sigv4a-signing-examples project on GitHub.

When to sign requests

When you write custom code that sends API requests to AWS, you must include code that signs the requests. You might write custom code because:

- You are working with a programming language for which there is no AWS SDK.
- You need complete control over how requests are sent to AWS.

While API requests authenticate access with AWS SigV4, AWS SDKs and the AWS CLI authenticate your requests by using the access keys that you provide. For more information about authenticating with AWS SDKs and the AWS CLI, see Additional resources.

Why requests are signed

The signing process helps secure requests in the following ways:

- **Verify the identity of the requester**

  Authenticated requests require a signature that you create by using your access keys (access key ID, secret access key). If you are using temporary security credentials, the signature calculations also require a security token. For more information, see AWS security credentials programmatic access.
- **Protect data in transit**

  To prevent tampering with a request while it's in transit, some of the request elements are used to calculate a hash (digest) of the request, and the resulting hash value is included as part of the request. When an AWS service receives the request, it uses the same information to calculate a hash and matches it against the hash value in your request. If the values don't match, AWS denies the request.
- **Protect against potential replay attacks**

  In most cases, a request must reach AWS within five minutes of the time stamp in the request. Otherwise, AWS denies the request.

AWS SigV4 can be expressed in the HTTP Authorization header or as a query string in the URL. For more information, see Authentication methods.

Additional resources

- For more information about the SigV4 signing process for different services, see Request signature examples.
- To configure credentials for programmatic access for the AWS CLI, see Authentication and access credentials in the *AWS Command Line Interface User Guide*.
- The AWS SDKs include source code on GitHub for signing AWS API requests. For code samples, see Example projects in AWS samples repository.
  - AWS SDK for .NET – AWSSigV4Signer.cs

- AWS SDK for C++ – AWSAuthV4Signer.cpp
- AWS SDK for Go – sigv4.go
- AWS SDK for Java – BaseAws4Signer.java
- AWS SDK for JavaScript – signature-v4
- AWS SDK for PHP – SignatureV4.php
- AWS SDK for Python (Boto) – signers.py
- AWS SDK for Ruby – signer.rb

- If you encounter errors when signing requests, see Troubleshoot Signature Version 4 signing for AWS API requests.

Elements of an AWS API request signature

> **Important**
>
> Unless you are using the AWS SDKs or CLI, you must write code to calculate signatures that provide authentication information in your requests. Signature calculation in AWS Signature Version 4 can be a complex undertaking, and we recommend that you use the AWS SDKs or CLI whenever possible.

Each HTTP/HTTPS request that uses Signature Version 4 signing must contain these elements.

Elements

- Endpoint specification
- Action
- Action parameters
- Date
- Authentication information

Endpoint specification

Specifies the DNS name of the endpoint to which you send the request. This name usually contains the service code and the Region. For example, the endpoint for Amazon DynamoDB in the `us-east-1` Region is `dynamodb.us-east-1.amazonaws.com`.

For HTTP/1.1 requests, you must include the `Host` header. For HTTP/2 requests, you can include the `:authority` header or the `Host` header. Use only the `:authority` header for compliance with the HTTP/2 specification. Not all services support HTTP/2 requests.

For the endpoints supported by each service, see Service endpoints and quotas in the *AWS General Reference*.

Action

Specifies an API action for the service. For example, the DynamoDB `CreateTable` action or the Amazon EC2 `DescribeInstances` action.

For the actions supported by each service, see the Service Authorization Reference.

Action parameters

Specifies the parameters for the action specified in the request. Each AWS API action has a set of required and optional parameters. The API version is usually a required parameter.

For the parameters supported by an API action, see the API Reference for the service.

Date

Specifies the date and time of the request. Including the date and time in a request helps prevent third parties from intercepting your request and resubmitting it later. The date that you specify in the credential scope must match the date of your request.

The time stamp must be in UTC and use the following ISO 8601 format: *YYYYMMDD*T*HHMMSS*Z. For example, `20220830T123600Z`. Do not include milliseconds in the time stamp.

You can use a `date` or an `x-amz-date` header, or include `x-amz-date` as a query parameter. If we can't find an `x-amz-date` header, then we look for a `date` header.

Authentication information

Each request that you send must include the following information. AWS uses this information to ensure the validity and authenticity of the request.

- Algorithm – The algorithm that you're using as part of the signing process.
  - SigV4 – Use `AWS4-HMAC-SHA256` to specify Signature Version 4 with the `HMAC-SHA256` hash algorithm.

  - SigV4a – Use `AWS4-ECDSA-P256-SHA256` to specify the `ECDSA-P256-SHA-256` hash algorithm.
- Credential – A string that is formed by concatenating your access key ID and your credential scope components.
  - SigV4 – Credential scope includes your access key ID, the date in *YYYYMMDD* format, the Region code, the service code, and the `aws4_request` termination string, separated by slashes (/). The Region code, service code, and termination string must use lowercase characters.

    ```
    AKIAIOSFODNN7EXAMPLE/YYYYMMDD/region/service/aws4_request
    ```
  - SigV4a – Credential scope includes the date in YYYYMMDD format, the service name, and the `aws4_request` termination string, separated by slashes (/). Note that credential scope does not include the region as the region is covered in a separate header `X-Amz-Region-Set`.

    ```
    AKIAIOSFODNN7EXAMPLE/YYYYMMDD/service/aws4_request
    ```
- Signed headers – The HTTP headers to include in the signature, separated by semicolons (;). For example, `host;x-amz-date`.

  For SigV4a, you must include a region set header that specifies the set of regions the request will be valid in. The header `X-Amz-Region-Set` is specified as a list of comma separated values. The following example shows a region header that allows a request to be made in both us-east-1 and us-west-1 regions.

  ```
  X-Amz-Region-Set=us-east-1,us-west-1
  ```

  You can use wildcards (*) in regions to specify multiple regions. In the following example, the header allows a request to be made in both us-west-1 and us-west-2.

  ```
  X-Amz-Region-Set=us-west-*
  ```
- Signature – A hexadecimal-encoded string that represents the calculated signature. You must calculate the signature using the algorithm that you specified in the `Algorithm` parameter.

For more information, see Authentication methods

Authentication methods

> **Important**
>
> Unless you are using the AWS SDKs or CLI, you must write code to calculate signatures that provide authentication information in your requests. Signature calculation in AWS Signature Version 4 can be a complex undertaking, and we recommend that you use the AWS SDKs or CLI whenever possible.

You can express authentication information by using one of the following methods.

HTTP authorization header

The HTTP `Authorization` header is the most common method of authenticating a request. All REST API operations (except for browser-based uploads using POST requests) require this header.

The following examples show the `Authorization` header value for SigV4 and SigV4a. Line breaks are added to this example for readability. In your code, the header must be a continuous string. There is no comma between the algorithm and Credential, but the other elements must be separated by commas.

Example SigV4

```
Authorization: AWS4-HMAC-SHA256
Credential=AKIAIOSFODNN7EXAMPLE/20130524/us-east-1/s3/aws4_request,
SignedHeaders=host;range;x-amz-date,
Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024
```

Example SigV4a

```
Authorization: AWS4-ECDSA-P256-SHA256
Credential=AKIAIOSFODNN7EXAMPLE/20130524/s3/aws4_request,
SignedHeaders=host;range;x-amz-date;x-amz-region-set,
Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024
```

The following table describes the various components of the Authorization header value in the preceding example:

Component	Description
Authorization	The algorithm that was used to calculate the signature. • SigV4 – Use `AWS4-HMAC-SHA256` . This string identifies AWS SigV4 (`AWS4`) and the `HMAC-SHA256` algorithm. • SigV4a – Use `AWS4-ECDSA-P256-SHA256` . This string identifies AWS SigV4 (`AWS4`) and the `ECDSA-P256-SHA-256` algorithm.

Credential	Your access key ID and the scope information.
	• SigV4 – Include the date, Region, and service that were used to calculate the signature. This string has the following form:
	`<your-access-key-id>/<date>/ <aws-region>/<aws-service>/ aws4_request`
	• SigV4a – Include the date and service that were used to calculate the signature. This string has the following form:
	`<your-access-key-id>/<date>/ <aws-service>/aws4_request`
	The <date> value is specified using YYYYMMDD format. <aws-service> value is S3 when sending request to Amazon S3.

SignedHeaders	A semicolon-separated list of request headers that you used to compute Signature. The list includes header names only, and the header

Component	Description
	names must be in lowercase. For example: `host;range;x-amz-date` For SigV4a, you must include a region set header that specifies the set of regions the request will be valid in. The header X-Amz-Region-Set is specified as a list of comma separated values.
Signature	The 256-bit signature expressed as 64 lowercase hexadecimal characters. For example:`fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024` Note that the signature calculations vary depending on the option you choose to transfer the payload.

Query string parameters

You can use a query string to express a request entirely in a URL. In this case, you use query parameters to provide request information, including the authentication information. Because the request signature is part of the URL, this type of URL is often referred to as a presigned URL. You can use presigned URLs to embed clickable links in HTML, which can be valid for up to seven days. For more information, see Authenticating Requests: Using Query Parameters (AWS Signature Version 4) in the *Amazon S3 API Reference*.

The following examples show presigned URLs for SigV4 and SigV4a. Line breaks are added to this example for readability:

Example SigV4

```
https://s3.amazonaws.com/amzn-s3-demo-bucket/test.txt ?
X-Amz-Algorithm=AWS4-HMAC-SHA256 &
X-Amz-Credential=<your-access-key-id>/20130721/<region>/s3/aws4_request &
X-Amz-Date=20130721T201207Z &
```

```
X-Amz-Expires=86400 &
X-Amz-SignedHeaders=host &X-Amz-Signature=<signature-value>
```

Example SigV4a

```
https://s3.amazonaws.com/amzn-s3-demo-bucket/test.txt ?
X-Amz-Algorithm=AWS4-ECDSA-P256-SHA256 &
X-Amz-Credential=<your-access-key-id>/20240721/s3/aws4_request &
X-amz-Region-Set=<regionset> &
X-Amz-Date=20240721T201207Z &
X-Amz-Expires=86400 &
X-Amz-SignedHeaders=host;x-amz-region-set &
X-Amz-Signature=<signature-value>
```

Note

The `X-Amz-Credential` value in the URL shows the "/" character only for readability. In practice, it should be encoded as %2F. For example: `&X-Amz-Credential=<your-access-key-id>%2F20130721%2Fus-east-1%2Fs3%2Faws4_request`

The following table describes the query parameters in the URL that provide authentication information.

Query string parameter name	Description
X-Amz-Algorithm	The version of AWS signature and the algorithm that you used to calculate the signature. • SigV4 – Use `AWS4-HMAC-SHA256` . This string identifies AWS SigV4 (`AWS4`) and the `HMAC-SHA256` algorithm. • SigV4a – Use `AWS4-ECDSA-P256-SHA256` . This string identifies AWS SigV4 (`AWS4`) and the `ECDSA-P256-SHA-256` algorithm.

Query string parameter name	Description
X-Amz-Credential	In addition to your access key ID, this parameter also provides scope for which the signature is valid. This value must match the scope you use in signature calculations, discussed in the following section. • SigV4 – The general form for this parameter value is as follows: `<your-access-key-id>/<date>/<AWS Region>/<AWS-service>/aws4_request` For example: `AKIAIOSFODNN7EXAMPLE/20130721/us-east-1/s3/aws4_request` • SigV4a – The general form for this parameter value is as follows: `<your-access-key-id>/<date>/<AWS-service>/aws4_request` For example: `AKIAIOSFODNN7EXAMPLE/20130721/s3/aws4_request` The region for SigV4a is defined in the region set header `X-Amz-Region-Set` . For a list of AWS regional strings, see Regional Endpoints in the *AWS General Reference*.
X-Amz-Region-Set	The set of regions the request will be valid in. The header x-amz-region-set is specified as a list of comma separated values.

Query string parameter name	Description
X-Amz-Date	The date and time format must follow the ISO 8601 standard, and must be formatted with the `yyyyMMddTHHmmssZ` format. For example if the date and time was "08/01/2016 15:32:41.982-700" then it must first be converted to UTC (Coordinated Universal Time) and then submitted as "20160801T223241Z".
X-Amz-Expires	Provides the time period, in seconds, for which the generated presigned URL is valid. For example, 86400 (24 hours). This value is an integer. The minimum value you can set is 1, and the maximum is 604800 (seven days).A presigned URL can be valid for a maximum of seven days because the signing key you use in signature calculation is valid for up to seven days.
X-Amz-SignedHeaders	Lists the headers that you used to calculate the signature. The following headers are required in the signature calculations: • The HTTP host header. • Any x-amz-* headers that you plan to add to the request. • For SigV4a, X-Amz-Region-Set is required to specify the regions in which the request can be made. For added security, you should sign all the request headers that you plan to include in your request.

Query string parameter name	Description
X-Amz-Signature	Provides the signature to authenticate your request. This signature must match the signature the service calculates; otherwise, the service denies the request. For example, `733255ef022bec3f2a8701cd61d4b371f3f28c9f193a1f02279211d48d5193d7` Signature calculations are described in the following section.
X-Amz-Security-Token	Optional credential parameter if using credentials sourced from the STS service.

Create a signed AWS API request

Important

If you use an AWS SDK (see Sample Code and Libraries) or AWS Command Line Interface (AWS CLI) tool to send API requests to AWS, you can skip this section because the SDK and CLI clients authenticate your requests by using the access keys that you provide. Unless you have a good reason not to, we recommend that you always use an SDK or the CLI.
In Regions that support multiple signature versions, manually signing requests means you must specify which signature version is used. When you supply requests to Multi-Region Access Points, SDKs and the CLI automatically switch to using Signature Version 4A without additional configuration.

You can use the AWS SigV4 signing protocol to create a signed request for AWS API requests.

1. Creating a canonical request based on the request details.
2. Calculating a signature using your AWS credentials.
3. Adding this signature to the request as an Authorization header.

AWS then replicates this process and verifies the signature, granting or denying access accordingly.

To see how you can use AWS SigV4 to sign API requests, see Request signature examples.

The following table describes the functions that are used in the process of creating a signed request. You need to implement code for these functions. For more information, see the code examples in the AWS SDKs.

Function	Description
`Lowercase()`	Convert the string to lowercase.
`Hex()`	Lowercase base 16 encoding.
`SHA256Hash()`	Secure Hash Algorithm (SHA) cryptographic hash function.
`HMAC-SHA256()`	Computes HMAC by using the SHA256 algorithm with the signing key provided. This is the final signature when you sign with SigV4.
`ECDSA-Sign`	Elliptic Curve Digital Signature Algorithm (ECDSA) signature computed by using asymmetric signatures based on public-private key cryptography.
`KDF(K, Label, Context, L)`	A NIST SP800-108 KDF in Counter Mode using the PRF function HMAC-SHA256 as defined in NIST SP 800-108r1.
`Oct2Int(byte[ ])`	An octet to integer function as described in ANSI X9.62.
`Trim()`	Remove any leading or trailing whitespace.
`UriEncode()`	URI encode every byte. UriEncode() must enforce the following rules:

Function	Description
	• URI encode every byte except the unreserved characters: 'A'-'Z', 'a'-'z', '0'-'9', '-', '.', '_', and '~'. • The space character is a reserved character and must be encoded as "%20" (and not as "+"). • Each URI encoded byte is formed by a '%' and the two-digit hexadecimal value of the byte. • Letters in the hexadecimal value must be uppercase, for example "%1A". • Encode the forward slash character, '/', everywhere except in the object key name. For example, if the object key name is `photos/Jan/sample.jpg` , the forward slash in the key name is not encoded. **Important** The standard UriEncode functions provided by your development platform might not work because of differences in implementation and related ambiguity in the underlying RFCs. We recommend that you write your own custom UriEncode function to make sure that your encoding will work. To see an example of a UriEncode function in Java, see Java Utilities on the GitHub website.

> **Note**
>
> When signing your requests, you can use either AWS SigV4 or AWS SigV4a. The key difference between the two is determined by how the signature is calculated. With SigV4a, the region set is included in the string to sign, but is not part of the credential derivation step.

Signing requests with temporary security credentials

Instead of using long-term credentials to sign a request, you can use temporary security credentials provided by AWS Security Token Service (AWS STS).

When you use temporary security credentials, you must add `X-Amz-Security-Token` to the Authorization header or include it in the query string to hold the session token. Some services require that you add `X-Amz-Security-Token` to the canonical request. Other services require only that you add `X-Amz-Security-Token` at the end, after you calculate the signature. Check the documentation for each AWS service for specific requirements.

Summary of signing steps

Create a canonical request

Arrange the contents of your request (host, action, headers, etc.) into a standard canonical format. The canonical request is one of the inputs used to create the string to sign. For details on creating the canonical request, see Elements of an AWS API request signature.

Create a hash of the canonical request

Hash the canonical request using the same algorithm that you used to create the hash of the payload. The hash of the canonical request is a string of lowercase hexadecimal characters.

Create a string to sign

Create a string to sign with the canonical request and extra information such as the algorithm, request date, credential scope, and the hash of the canonical request.

Derive a signing key

Use the secret access key to derive the key used to sign the request.

Calculate the signature

Perform a keyed hash operation on the string to sign using the derived signing key as the hash key.

Add the signature to the request

Add the calculated signature to an HTTP header or to the query string of the request.

Create a canonical request

To create a canonical request, concatenate the following strings, separated by newline characters. This helps make sure that the signature that you calculate can match the signature that AWS calculates.

```
<HTTPMethod>\n
<CanonicalURI>\n
<CanonicalQueryString>\n
<CanonicalHeaders>\n
<SignedHeaders>\n
<HashedPayload>
```

- *HTTPMethod* – The HTTP method, such as GET, PUT, HEAD, and DELETE.
- *CanonicalUri* – The URI-encoded version of the absolute path component URI, starting with the / that follows the domain name and up to the end of the string or to the question mark character (?) if you have query string parameters. If the absolute path is empty, use a forward slash character (/). The URI in the following example, /amzn-s3-demo-bucket/myphoto.jpg, is the absolute path and you don't encode the / in the absolute path:

  ```
  http://s3.amazonaws.com/amzn-s3-demo-bucket/myphoto.jpg
  ```

- *CanonicalQueryString* – The URI-encoded query string parameters. You URI-encode each name and value individually. You must also sort the parameters in the canonical query string alphabetically by key name. The sorting occurs after encoding. The query string in the following URI example is:

  ```
  http://s3.amazonaws.com/amzn-s3-demo-bucket?prefix=somePrefix&marker=someMarker&max-keys=2
  ```

  The canonical query string is as follows (line breaks are added to this example for readability):

  ```
  UriEncode("marker")+"="+UriEncode("someMarker")+"&"+
  UriEncode("max-keys")+"="+UriEncode("20") + "&" +
  ```

```
UriEncode("prefix")+"="+UriEncode("somePrefix")
```

When a request targets a subresource, the corresponding query parameter value will be an empty string (""). For example, the following URI identifies the `ACL` subresource on the `amzn-s3-demo-bucket` bucket:

```
http://s3.amazonaws.com/amzn-s3-demo-bucket?acl
```

In this case, the CanonicalQueryString would be:

```
UriEncode("acl") + "=" + ""
```

If the URI does not include a `?`, there is no query string in the request, and you set the canonical query string to an empty string (""). You will still need to include the newline character ("\n").

- *`CanonicalHeaders`* – A list of request headers with their values. Individual header name and value pairs are separated by the newline character ("\n"). The following is an example of a CanonicalHeader:

```
Lowercase(<HeaderName1>)+":"+Trim(<value>)+"\n"
Lowercase(<HeaderName2>)+":"+Trim(<value>)+"\n"
...
Lowercase(<HeaderNameN>)+":"+Trim(<value>)+"\n"
```

CanonicalHeaders list must include the following:

- HTTP `host` header.
- If the `Content-Type` header is present in the request, you must add it to the *`CanonicalHeaders`* list.
- Any `x-amz-*` headers that you plan to include in your request must also be added. For example, if you are using temporary security credentials, you need to include `x-amz-security-token` in your request. You must add this header in the list of *`CanonicalHeaders`*.
- For SigV4a, you must include a region set header that specifies the set of regions the request will be valid in. The header `X-Amz-Region-Set` is specified as a list of comma separated values. The following example shows a region header that allows a request to be made in both us-east-1 and us-west-1 regions.

```
X-Amz-Region-Set=us-east-1,us-west-1
```

You can use wildcards (*) in regions to specify multiple regions. In the following example, the header allows a request to be made in both us-west-1 and us-west-2.

```
X-Amz-Region-Set=us-west-*
```

> **Note**
>
> The `x-amz-content-sha256` header is required for Amazon S3 AWS requests. It provides a hash of the request payload. If there is no payload, you must provide the hash of an empty string.

Each header name must:

- use lowercase characters.
- appear in alphabetical order.
- be followed by a colon (`:`).

For values, you must:

- trim any leading or trailing spaces.
- convert sequential spaces to a single space.
- separate the values for a multi-value header using commas.
- You must include the host header (HTTP/1.1) or the :authority header (HTTP/2), and any `x-amz-*` headers in the signature. You can optionally include other standard headers in the signature, such as content-type.

The `Lowercase()` and `Trim()` functions used in this example are described in the preceding section.

The following is an example `CanonicalHeaders` string. The header names are in lowercase and sorted.

```
host:s3.amazonaws.com
x-amz-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
x-amz-date:20130708T220855Z
```

> **Note**
>
> For the purpose of calculating an authorization signature, only the host and any `x-amz-*` headers are required; however, in order to prevent data tampering, you should consider including additional headers in the signature calculation.
> Do not include hop-by-hop headers that are frequently altered during transit across a complex system. This includes all volatile transport headers that are mutated by proxies, load balancers, and the nodes in a distributed system, including `connection`, `x-amzn-trace-id`, `user-agent`, `keep-alive`, `transfer-encoding`, `TE`, `trailer`, `upgrade`, `proxy-authorization`, and `proxy-authenticate`.

- *`SignedHeaders`* – An alphabetically sorted, semicolon-separated list of lowercase request header names. The request headers in the list are the same headers that you included in the `CanonicalHeaders` string. For the previous example, the value of *`SignedHeaders`* would be as follows:

```
host;x-amz-content-sha256;x-amz-date
```

- *`HashedPayload`* – A string created using the payload in the body of the HTTP request as input to a hash function. This string uses lowercase hexadecimal characters.

```
Hex(SHA256Hash(<payload>))
```

If there is no payload in the request, you compute a hash of the empty string, such as when you retrieve an object by using a `GET` request, there is nothing in the payload.

```
Hex(SHA256Hash(""))
```

> **Note**
>
> For Amazon S3, include the literal string `UNSIGNED-PAYLOAD` when constructing a canonical request, and set the same value as the `x-amz-content-sha256` header value when sending the request.
> `Hex(SHA256Hash("UNSIGNED-PAYLOAD"))`

Create a hash of the canonical request

Create a hash (digest) of the canonical request using the same algorithm that you used to create the hash of the payload. The hash of the canonical request is a string of lowercase hexadecimal characters.

Create a string to sign

To create a string to sign, concatenate the following strings, separated by newline characters. Do not end this string with a newline character.

```
Algorithm \n
RequestDateTime \n
CredentialScope  \n
HashedCanonicalRequest
```

- *Algorithm* – The algorithm used to create the hash of the canonical request.
  - SigV4 – Use `AWS4-HMAC-SHA256` to specify the `HMAC-SHA256` hash algorithm.
  - SigV4a – Use `AWS4-ECDSA-P256-SHA256` to specify the `ECDSA-P256-SHA-256` hash algorithm.
- *RequestDateTime* – The date and time used in the credential scope. This value is the current UTC time in ISO 8601 format (for example, `20130524T000000Z`).
- *CredentialScope* – The credential scope, which restricts the resulting signature to the specified Region and service.
  - SigV4 – Credentials include your access key ID, the date in `YYYYMMDD` format, the Region code, the service code, and the `aws4_request` termination string, separated by slashes (/). The Region code, service code, and termination string must use lowercase characters. The string has the following format: `YYYYMMDD/region/service/aws4_request`.
  - SigV4a – Credentials include the date in `YYYYMMDD` format, the service name, and the `aws4_request` termination string, separated by slashes (/). Note that credential scope does not include the region as the region is covered in a separate header `X-Amz-Region-Set`. The string has the following format: `YYYYMMDD/service/aws4_request`.
- *HashedCanonicalRequest* – The hash of the canonical request, calculated in the previous step.

The following is an example string to sign.

```
"<Algorithm>" + "\n" +
timeStampISO8601Format + "\n" +
<Scope> + "\n" +
Hex(<Algorithm>(<CanonicalRequest>))
```

Derive a signing key

To derive a signing key, choose one of the following processes to compute a signing key for either SigV4 or SigV4a.

Deriving a signing key for SigV4

To derive a signing key for SigV4, perform a succession of keyed hash operations (HMAC) on the request date, Region, and service, with your AWS secret access key as the key for the initial hashing operation.

For each step, call the hash function with the required key and data. The result of each call to the hash function becomes the input for the next call to the hash function.

The following example shows how you derive the `SigningKey` used in the next section of this procedure, showing the order in which your input is concatenated and hashed. `HMAC-SHA256` is the hash function used to hash the data as shown.

```
DateKey = HMAC-SHA256("AWS4"+"<SecretAccessKey>", "<YYYYMMDD>")
DateRegionKey = HMAC-SHA256(<DateKey>, "<aws-region>")
DateRegionServiceKey = HMAC-SHA256(<DateRegionKey>, "<aws-service>")
SigningKey = HMAC-SHA256(<DateRegionServiceKey>, "aws4_request")
```

Required input

- `Key` – A string that contains your secret access key.
- `Date` – A string that contains the date used in the credential scope, in the format *YYYYMMDD*.
- `Region` – A string that contains the Region code (for example, `us-east-1`).

  For a list of Region strings, see Regional Endpoints in the *AWS General Reference*.
- `Service` – A string that contains the service code (for example, `ec2`).
- The string to sign that you created in the previous step.

To derive a signing key for SigV4

1. Concatenate "AWS4" and the secret access key. Call the hash function with the concatenated string as the key and the date string as the data.

```
DateKey = hash("AWS4" + Key, Date)
```

2. Call the hash function with the result of the previous call as the key and the Region string as the data.

```
DateRegionKey = hash(kDate, Region)
```

3. Call the hash function with the result of the previous call as the key and the service string as the data.

   The service code is defined by the service. You can use get-products in the *AWS Pricing CLI* to return the service code for a service.

```
DateRegionServiceKey = hash(kRegion, Service)
```

4. Call the hash function with the result of the previous call as the key and "aws4_request" as the data.

```
SigningKey = hash(kService, "aws4_request")
```

Deriving a signing key for SigV4a

To create a signing key for SigV4a, use the following process to derive a key pair from the secret access key. For an example of an implementation of this derivation, see the C99 library implementation of AWS client-side authentication

```
n = [NIST P-256 elliptic curve group order]
G = [NIST P-256 elliptic curve base point]
label = "AWS4-ECDSA-P256-SHA256"

akid = [AWS access key ID as a UTF8 string]
sk = [AWS secret access Key as a UTF8 Base64 string]

input_key = "AWS4A" || sk
count = 1
```

```
while (counter != 255) {
  context = akid || counter // note: counter is one byte
  key = KDF(input_key, label, context, 256)
  c = Oct2Int(key)
  if (c > n - 2) {
    counter++
  } else {
    k = c + 1   // private key
    Q = k * G   // public key
  }
}

if (c < 255) {
  return [k, Q]
} else {
  return FAILURE
}
```

Calculate the signature

After you have derived the signing key, calculate the signature to add to your request. This procedure varies based on the signature version you use.

To calculate a signature for SigV4

1. Call the hash function with the result of the previous call as the key and the **string to sign** as the data. Use the derived signing key as the hash key for this operation. The result is the signature as a binary value.

   ```
   signature = hash(SigningKey, string-to-sign)
   ```

2. Convert the signature from binary to hexadecimal representation, in lowercase characters.

To calculate a signature for SigV4a

1. Using the digital signing algorithm (ECDSA P-256), sign the **string to sign** you created in the previous step. The key used for this signature is the private asymmetric key derived from the secret access key as described above.

   ```
   signature = base16(ECDSA-Sign(k, string-to-sign))
   ```

2. Convert the signature from binary to hexadecimal representation, in lowercase characters.

Add the signature to the request

Add the calculated signature to your request.

Example: Authorization header

SigV4

The following example shows an `Authorization` header for the `DescribeInstances` action using AWS SigV4. For readability, this example is formatted with line breaks. In your code, this must be a continuous string. There is no comma between the algorithm and `Credential`. However, the other elements must be separated by commas.

```
Authorization: AWS4-HMAC-SHA256
Credential=AKIAIOSFODNN7EXAMPLE/20220830/us-east-1/ec2/aws4_request,
SignedHeaders=host;x-amz-date,
Signature=calculated-signature
```

SigV4a

The following example shows an Authorization header for the `CreateBucket` action using AWS SigV4a. For readability, this example is formatted with line breaks. In your code, this must be a continuous string. There is no comma between the algorithm and Credential. However, the other elements must be separated by commas.

```
Authorization: AWS4-ECDSA-P256-SHA256
Credential=AKIAIOSFODNN7EXAMPLE/20220830/s3/aws4_request,
SignedHeaders=host;x-amz-date;x-amz-region-set,
Signature=calculated-signature
```

Example: Request with authentication parameters in the query string

SigV4

The following example shows a query for the `DescribeInstances` action using AWS SigV4 that includes the authentication information. For readability, this example is formatted with line breaks and is not URL encoded. In your code, the query string must be a continuous string that is URL encoded.

```
https://ec2.amazonaws.com/?
Action=DescribeInstances&
Version=2016-11-15&
X-Amz-Algorithm=AWS4-HMAC-SHA256&
X-Amz-Credential=AKIAIOSFODNN7EXAMPLE/20220830/us-east-1/ec2/aws4_request&
X-Amz-Date=20220830T123600Z&
X-Amz-SignedHeaders=host;x-amz-date&
X-Amz-Signature=calculated-signature
```

SigV4a

The following example shows a query for the `CreateBucket` action using AWS SigV4a that includes the authentication information. For readability, this example is formatted with line breaks and is not URL encoded. In your code, the query string must be a continuous string that is URL encoded.

```
https://ec2.amazonaws.com/?
Action=CreateBucket&
Version=2016-11-15&
X-Amz-Algorithm=AWS4-ECDSA-P256-SHA256&
X-Amz-Credential=AKIAIOSFODNN7EXAMPLE/20220830/s3/aws4_request&
X-Amz-Region-Set=us-west-1&
X-Amz-Date=20220830T123600Z&
X-Amz-SignedHeaders=host;x-amz-date;x-amz-region-set&
X-Amz-Signature=calculated-signature
```

Request signature examples

The following examples of AWS signing requests show you how you can use SigV4 to sign requests sent without the AWS SDK or AWS command line tool.

Browser based Amazon S3 upload using HTTP POST

Authenticating Requests: Browser-Based Uploads describes the signature and relevant information that Amazon S3 uses to calculate the signature upon receiving the request.

Example: Browser-Based Upload using HTTP POST (Using AWS Signature Version 4) provides more information with a sample POST policy and a form that you can use to upload a file. The example policy and fictitious credentials show you the workflow and resulting signature and policy hash.

VPC Lattice authenticated requests

Examples for Signature Version 4 (SigV4) authenticated requests provides Python and Java examples showing how you can perform request signing with and without custom interceptors.

Using Signature Version 4 with Amazon Translate

Live Translations in the Metaverse shows how to build an application that produces a near real-time translation solution. This speech-to-speech translator solution uses AWS SigV4 in event stream encoding to produce real-time transcriptions.

Using Signature Version 4 with Neptune

Example: Connecting to Neptune Using Python with Signature Version 4 Signing shows how to make signed requests to Neptune using Python. This example includes variations for using an access key or temporary credentials.

Signing HTTP requests to Amazon Glacier

Example Signature Calculation for Streaming API walks you through the details of creating a signature for Upload Archive (POST archive), one of the two streaming APIs in Amazon Glacier.

Making HTTP Requests to Amazon SWF

Making HTTP Requests to Amazon SWF shows the header contents for a JSON request to Amazon SWF.

Signature calculation for streaming APIs in Amazon OpenSearch Service

Signing an Amazon OpenSearch Service search request with AWS SDK for PHP Version 3 includes an example of how to send signed HTTP requests to Amazon OpenSearch Service.

Example projects in AWS samples repository

The following example projects show how to sign requests to make Rest API requests to AWS services with common languages such as Python, Node.js, Java, C#, Go and Rust.

Signature Version 4a projects

The sigv4-signing-examples project provides examples of how to sign requests with SigV4 to make Rest API requests to AWS services with common languages such as Python, Node.js, Java, C#, Go and Rust.

The sigv4a-signing-examples project provides examples for signing multi-Region API requests, for example Multi-Region Access Points in Amazon S3.

Publish to AWS IoT Core

Python code to publish to AWS IoT Core using HTTPs protocol provides guidance on how to publish messages to AWS IoT Core using HTTPS protocol and AWS SigV4 authentication. It has two reference implementations - one in Python and other in NodeJs.

.Net Framework application to publish to AWS IoT Core using HTTPs protocol provides guidance on how to publish messages to AWS IoT Core using HTTPS protocol and AWS SigV4 authentication. This project also includes a .NET core equivalent implementation.

Troubleshoot Signature Version 4 signing for AWS API requests

> **Important**
>
> Unless you are using the AWS SDKs or CLI, you must write code to calculate signatures that provide authentication information in your requests. SigV4 signature calculation can be a complex undertaking, and we recommend that you use the AWS SDKs or CLI whenever possible.

When you develop code that creates a signed request, you might receive HTTP 403 `SignatureDoesNotMatch` from AWS services. These errors mean that the signature value in your HTTP request to AWS did not match the signature that the AWS service calculated. HTTP 401 `Unauthorized` errors return when permissions do not allow the caller to make the request.

API requests might return an error if:

- The API request isn't signed and the API request uses IAM authentication.
- The IAM credentials used to sign the request are incorrect or don't have permissions to invoke the API.
- The signature of the signed API request doesn't match the signature that the AWS service calculated.
- The API request header is incorrect.

> **Note**
>
> Update your signing protocol from AWS Signature version 2 (SigV2) to AWS Signature version 4 (SigV4) before exploring other error solutions. Services, such as Amazon S3, and Regions no longer support SigV2 signing.

Possible causes

- Credential errors
- Canonical request and signing string errors
- Credential scope errors
- Key signing errors

Credential errors

Make sure that the API request is signed with SigV4. If the API request isn't signed, then you might receive the error: `Missing Authentication Token`. Add the missing signature and resend the request.

Verify that the authentication credentials for the access key and secret key are correct. If the access key is incorrect, then you might receive the error: `Unauthorized`. Make sure the entity used to sign the request is authorized to make the request. For details, see Troubleshoot access denied error messages.

Canonical request and signing string errors

If you incorrectly calculated the canonical request in Create a hash of the canonical request or Create a string to sign, the signature verification step performed by the service fails with the error message:

```
The request signature we calculated does not match the signature you provided
```

When the AWS service receives a signed request, it recalculates the signature. If there are differences in the values, then the signatures don’t match. Compare the canonical request and string to your signed request with the value in the error message. Modify the signing process if there are any differences.

> **Note**
>
> You can also verify that you didn't send the request through a proxy that modifies the headers or the request.

Example canonical request

```
GET                                                     -------- HTTP method
/                                                       -------- Path. For API stage
 endpoint, it should be /{stage-name}/{resource-path}
                                                        -------- Query string key-
value pair. Leave it blank if the request doesn't have a query string.
content-type:application/json                           -------- Header key-value
 pair. One header per line.
host:0123456789.execute-api.us-east-1.amazonaws.com     -------- Host and x-amz-date
 are required headers for all signed requests.
x-amz-date:20220806T024003Z

content-type;host;x-amz-date                            -------- A list of signed
 headers
d167e99c53f15b0c105101d468ae35a3dc9187839ca081095e340f3649a04501        -------- Hash
 of the payload
```

To verify that the secret key matches the access key ID, you can test them with a known working implementation. For example, use an AWS SDK or the AWS CLI to make a request to AWS.

API request header

When the authorization header is empty, the credential key or signature is missing or incorrect, the header doesn’t start with an algorithm name, or the key value pairs don’t include an equal sign, you receive one of the following errors:

- Authorization header cannot be empty.
- Authorization header requires 'Credential' parameter.
- Authorization header requires 'Signature' parameter.
- The signature contains an invalid key=value pair (missing equal-sign) in Authorization header.

Make sure that the SigV4 authorization header you added in Calculate the signature includes the correct credential key, and also includes the request date using either HTTP Date or the `x-amz-date` header.

If you received an IncompleteSignatureException error and the construction of the signature is correct, you can verify that the authorization header was not modified in transit to the AWS service by computing a SHA-256 hash and B64 encoding of the authorization header in your client-side request.

1. Get the authorization header you sent in the request. Your authorization header appears similar to the following example:

```
Authorization: AWS4-HMAC-SHA256
Credential=AKIAIOSFODNN7EXAMPLE/20130524/us-east-1/s3/aws4_request,
SignedHeaders=host;range;x-amz-date,
Signature=example-generated-signature
```

2. Compute a SHA-256 hash of the authorization header.

```
hashSHA256(rawAuthorizationHeader) = hashedAuthorizationHeader
```

3. Encode the hashed authorization header to Base64 format.

```
base64(hashedAuthorizationHeader) = encodedHashedAuthorizationHeader
```

4. Compare the hashed and encoded string you just calculated against the string you received in your error message. Your error message should be similar to the following example:

```
com.amazon.coral.service#IncompleteSignatureException:
The signature contains an in-valid key=value pair (missing equal-sign)
in Authorization header (hashed with SHA-256 and encoded with Base64):
'9c574f83b4b950926da4a99c2b43418b3db8d97d571b5e18dd0e4f3c3ed1ed2c'.
```

- If the two hashes differ, then some part of the authorization header changed in transit. This change could be due to your network or client handlers attaching signed headers or changing the authorization header in some way.
- If the two hashes match, the authorization header you sent in the request matches what AWS received. Review the error message you received to determine if the issue is the result of incorrect credentials or signature. These errors are covered in the other sections on this page.

Credential scope errors

The credential scope you created in Create a string to sign restricts a signature to a specific date, Region, and service. This string has the following format:

```
YYYYMMDD/region/service/aws4_request
```

> **Note**
>
> If you are using SigV4a, the Region is not included in credential scope.

Date

If the credential scope does not specify the same date as the x-amz-date header, the signature verification step fails with the following error message:

```
Date in Credential scope does not match YYYYMMDD from ISO-8601 version of date from
 HTTP
```

If the request specifies a time in the future, the signature verification step fails with the following error message:

```
Signature not yet current: date is still later than date
```

If the request has expired, the signature verification step fails with the following error message:

```
Signature expired: date is now earlier than date
```

Region

If the credential scope does not specify the same Region as the request, the signature verification step fails with the following error message:

```
Credential should be scoped to a valid Region, not region-code
```

Service

If the credential scope does not specify the same service as the host header, the signature verification step fails with the following error message:

```
Credential should be scoped to correct service: 'service'
```

Termination string

If the credential scope does not end with aws4_request, the signature verification step fails with the following error message:

```
Credential should be scoped with a valid terminator: 'aws4_request'
```

Key signing errors

Errors that are caused by incorrect derivation of the signing key or improper use of cryptography are more difficult to troubleshoot. After you verify that the canonical string and the string to sign are correct, you can also check for one of the following issues:

- The secret access key does not match the access key ID that you specified.
- There is a problem with your key derivation code.

To verify that the secret key matches the access key ID, you can test them with a known working implementation. For example, use an AWS SDK or the AWS CLI to make a request to AWS. For examples, see Request signature examples

IAM JSON policy reference

This section presents detailed syntax, descriptions, and examples of the elements, variables, and evaluation logic of JSON policies in IAM. For more general information, see Overview of JSON policies.

This reference includes the following sections.

- IAM JSON policy element reference — Learn more about the elements that you can use when you create a policy. View additional policy examples and learn about conditions, supported data types, and how they are used in various services.
- Policy evaluation logic — This section describes AWS requests, how they are authenticated, and how AWS uses policies to determine access to resources.

- Grammar of the IAM JSON policy language — This section presents a formal grammar for the language that is used to create policies in IAM.
- AWS managed policies for job functions — This section lists all the AWS managed policies that directly map to common job functions in the IT industry. Use these policies to grant the permissions that are needed to carry out the tasks expected of someone in a specific job function. These policies consolidate permissions for many services into a single policy.
- AWS global condition context keys — This section includes a list of all the AWS global condition keys that you can use to limit permissions in an IAM policy.
- IAM and AWS STS condition context keys — This section includes a list of all the IAM and AWS STS condition keys that you can use to limit permissions in an IAM policy.
- Actions, Resources, and Condition Keys for AWS Services — This section presents a list of all the AWS API operations that you can use as permissions in an IAM policy. It also includes the service-specific condition keys that can be used to further refine the request.

IAM JSON policy element reference

JSON policy documents are made up of elements. The elements are listed here in the general order you use them in a policy. The order of the elements doesn't matter—for example, the `Resource` element can come before the `Action` element. You're not required to specify any `Condition` elements in the policy. To learn more about the general structure and purpose of a JSON policy document, see Overview of JSON policies.

Some JSON policy elements are mutually exclusive. This means that you cannot create a policy that uses both. For example, you cannot use both `Action` and `NotAction` in the same policy statement. Other pairs that are mutually exclusive include `Principal`/`NotPrincipal` and `Resource`/`NotResource`.

The details of what goes into a policy vary for each service, depending on what actions the service makes available, what types of resources it contains, and so on. When you're writing policies for a specific service, it's helpful to see examples of policies for that service. For a list of all the services that support IAM, and for links to the documentation in those services that discusses IAM and policies, see AWS services that work with IAM.

When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about

policy validation, see IAM policy validation. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation.

Topics

- IAM JSON policy elements: Version
- IAM JSON policy elements: Id
- IAM JSON policy elements: Statement
- IAM JSON policy elements: Sid
- IAM JSON policy elements: Effect
- AWS JSON policy elements: Principal
- AWS JSON policy elements: NotPrincipal
- IAM JSON policy elements: Action
- IAM JSON policy elements: NotAction
- IAM JSON policy elements: Resource
- IAM JSON policy elements: NotResource
- IAM JSON policy elements: Condition
- IAM policy elements: Variables and tags
- IAM JSON policy elements: Supported data types

IAM JSON policy elements: Version

Disambiguation note

This `Version` JSON policy element is different from a *policy version*. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you make changes to a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. If you were searching for information about the multiple version support available for managed policies, see the section called “Versioning IAM policies”.

The `Version` policy element specifies the language syntax rules that are to be used to process a policy. To use all of the available policy features, include the following `Version` element **outside** the `Statement` element in all of your policies.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    }
  ]
}
```

IAM supports the following `Version` element values:

- `2012-10-17`. This is the current version of the policy language, and you should always include a `Version` element and set it to `2012-10-17`. Otherwise, you cannot use features such as policy variables that were introduced with this version.
- `2008-10-17`. This was an earlier version of the policy language. You might see this version on older existing policies. Do not use this version for any new policies or when you update any existing policies. Newer features, such as policy variables, will not work with your policy. For example, variables such as `${aws:username}` aren't recognized as variables and are instead treated as literal strings in the policy.

IAM JSON policy elements: Id

The `Id` element specifies an optional identifier for the policy. The ID is used differently in different services. ID is allowed in resource-based policies, but not in identity-based policies.

For services that let you set an `ID` element, we recommend you use a UUID (GUID) for the value, or incorporate a UUID as part of the ID to ensure uniqueness.

JSON

```
{
  "Version":"2012-10-17",
  "Id": "cd3ad3d9-2776-4ef1-a904-4c229d1642ee",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    }
  ]
}
```

Note

Some AWS services (for example, Amazon SQS or Amazon SNS) might require this element and have uniqueness requirements for it. For service-specific information about writing policies, refer to the documentation for the service you're working with.

IAM JSON policy elements: Statement

The `Statement` element is the main element for a policy. This element is required. The `Statement` element can contain a single statement or an array of individual statements. Each individual statement block must be enclosed in curly braces { }. For multiple statements, the array must be enclosed in square brackets [ ].

```
"Statement": [{...},{...},{...}]
```

The following example shows a policy that contains an array of three statements inside a single `Statement` element. (The policy allows you to access your own "home folder" in the Amazon S3 console.) The policy includes the `aws:username` variable, which is replaced during policy evaluation with the user name from the request. For more information, see Introduction.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Condition": {"StringLike": {"s3:prefix": [
        "",
        "home/",
        "home/${aws:username}/"
      ]}}
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:username}",
        "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:username}/*"
      ]
    }
  ]
}
```

IAM JSON policy elements: Sid

You can provide a `Sid` (statement ID) as an optional identifier for the policy statement. You can assign a `Sid` value to each statement in a statement array. You can use the `Sid` value as a description for the policy statement. In services that let you specify an `ID` element, such as SQS and SNS, the `Sid` value is just a sub-ID of the policy document ID. In IAM, the `Sid` value must be unique within a JSON policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStatementID",
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    }
  ]
}
```

The `Sid` element supports ASCII uppercase letters (A-Z), lowercase letters (a-z), and numbers (0-9).

IAM does not expose the `Sid` in the IAM API. You can't retrieve a particular statement based on this ID.

> **Note**
>
> Some AWS services (for example, Amazon SQS or Amazon SNS) might require this element and have uniqueness requirements for it. For service-specific information about writing policies, refer to the documentation for the service you work with.

IAM JSON policy elements: Effect

The `Effect` element is required and specifies whether the statement results in an allow or an explicit deny. Valid values for `Effect` are `Allow` and `Deny`. The `Effect` value is case sensitive.

```
"Effect":"Allow"
```

By default, access to resources is denied. To allow access to a resource, you must set the `Effect` element to `Allow`. To override an allow (for example, to override an allow that is otherwise in force), you set the `Effect` element to `Deny`. For more information, see Policy evaluation logic.

AWS JSON policy elements: Principal

Use the `Principal` element in a resource-based JSON policy to specify the principal that is allowed or denied access to a resource.

You must use the `Principal` element in resource-based policies. Several services support resource-based policies, including IAM. The IAM resource-based policy type is a role trust policy. In IAM roles, use the `Principal` element in the role trust policy to specify who can assume the role. For cross-account access, you must specify the 12-digit identifier of the trusted account. To learn whether principals in accounts outside of your zone of trust (trusted organization or account) have access to assume your roles, see What is IAM Access Analyzer?.

> **Note**
>
> After you create the role, you can change the account to "*" to allow everyone to assume the role. If you do this, we strongly recommend that you limit who can access the role through other means, such as a `Condition` element that limits access to only certain IP addresses. Do not leave your role accessible to everyone!

Other examples of resources that support resource-based policies include an Amazon S3 bucket or an AWS KMS key.

You cannot use the `Principal` element in an identity-based policy. Identity-based policies are permissions policies that you attach to IAM identities (users, groups, or roles). In those cases, the principal is implicitly the identity where the policy is attached.

Topics

- How to specify a principal
- AWS account principals
- IAM role principals
- Role session principals
- OIDC federated principals
- SAML federated principals
- IAM user principals
- IAM Identity Center principals
- AWS STS federated user principals

- AWS service principals
- AWS service principals in opt-in Regions
- All principals
- More information

How to specify a principal

You specify a principal in the `Principal` element of a resource-based policy or in condition keys that support principals.

You can specify any of the following principals in a policy:

- AWS account and root user
- IAM roles
- Role sessions
- IAM users
- Federated user principals
- AWS services
- All principals

You cannot identify a user group as a principal in a policy (such as a resource-based policy) because groups relate to permissions, not authentication, and principals are authenticated IAM entities.

You can specify more than one principal for each of the principal types in following sections using an array. Arrays can take one or more values. When you specify more than one principal in an element, you grant permissions to each principal. This is a logical `OR` and not a logical `AND`, because you authenticate as one principal at a time. If you include more than one value, use square brackets (`[` and `]`) and comma-delimit each entry for the array. The following example policy defines permissions for the 123456789012 account or the 555555555555 account.

```
"Principal" : {
"AWS": [
  "123456789012",
  "555555555555"
  ]
}
```

> **Note**
>
> You cannot use a wildcard to match part of a principal name or ARN.

AWS account principals

You can specify AWS account identifiers in the `Principal` element of a resource-based policy or in condition keys that support principals. This delegates authority to the account. When you allow access to a different account, an administrator in that account must then grant access to an identity (IAM user or role) in that account. When you specify an AWS account, you can use the account ARN (arn:aws:iam::*`account-ID`*:root), or a shortened form that consists of the `"AWS":` prefix followed by the account ID.

For example, given an account ID of `123456789012`, you can use either of the following methods to specify that account in the `Principal` element:

```
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

```
"Principal": { "AWS": "123456789012" }
```

The account ARN and the shortened account ID behave the same way. Both delegate permissions to the account. Using the account ARN in the `Principal` element does not limit permissions to only the root user of the account.

> **Note**
>
> When you save a resource-based policy that includes the shortened account ID, the service might convert it to the principal ARN. This does not change the functionality of the policy.

Some AWS services support additional options for specifying an account principal. For example, Amazon S3 lets you specify a canonical user ID using the following format:

```
"Principal": { "CanonicalUser":
 "79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be" }
```

You can also specify more than one AWS account, (or canonical user ID) as a principal using an array. For example, you can specify a principal in a bucket policy using all three methods.

```
"Principal": {
  "AWS": [
    "arn:aws:iam::123456789012:root",
    "999999999999"
  ],
  "CanonicalUser": "79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be"
}
```

IAM role principals

You can specify IAM role principal ARNs in the `Principal` element of a resource-based policy or in condition keys that support principals. IAM roles are identities. In IAM, identities are resources to which you can assign permissions. Roles trust another authenticated identity to assume that role. This includes a principal in AWS or a user from an external identity provider (IdP). When a principal or identity assumes a role, they receive temporary security credentials with the assumed role's permissions. When they use those session credentials to perform operations in AWS, they become a *role session principal*.

When you specify a role principal in a resource-based policy, the effective permissions for the principal are limited by any policy types that limit permissions for the role. This includes session policies and permissions boundaries. For more information about how the effective permissions for a role session are evaluated, see Policy evaluation logic.

To specify the role ARN in the `Principal` element, use the following format:

```
"Principal": { "AWS": "arn:aws:iam::AWS-account-ID:role/role-name" }
```

Important

If your `Principal` element in a role trust policy contains an ARN that points to a specific IAM role, then that ARN transforms to the role's unique principal ID when you save the policy. This helps mitigate the risk of someone escalating their privileges by removing and recreating the role. You don't normally see this ID in the console, because IAM uses a reverse transformation back to the role ARN when the trust policy is displayed. However, if you delete the role, then you break the relationship. The policy no longer applies, even if you recreate the role because the new role has a new principal ID that does not match the ID stored in the trust policy. When this happens, the principal ID appears in resource-based policies because AWS can no longer map it back to a valid ARN.

The result is that if you delete and recreate a role referenced in a trust policy's `Principal` element, you must edit the role in the policy to replace the principal ID with the correct ARN. The ARN once again transforms into the role's new principal ID when you save the policy. For more information, see Understanding AWS's Handling of Deleted IAM roles in Policies.

Alternatively, you can specify the role principal as the principal in a resource-based policy or create a broad-permission policy that uses the `aws:PrincipalArn` condition key. When you use this key, the role session principal is granted the permissions based on the ARN of role that was assumed, and not the ARN of the resulting session. Because AWS does not convert condition key ARNs to IDs, permissions granted to the role ARN persist if you delete the role and then create a new role with the same name. Identity-based policy types, such as permissions boundaries or session policies, do not limit permissions granted using the `aws:PrincipalArn` condition key with a wildcard(*) in the `Principal` element, unless the identity-based policies contain an explicit deny.

Role session principals

You can specify role sessions in the `Principal` element of a resource-based policy or in condition keys that support principals. When a principal or identity assumes a role, they receive temporary security credentials with the assumed role's permissions. When they use those session credentials to perform operations in AWS, they become a *role session principal*.

The format that you use for a role session principal depends on the AWS STS operation that was used to assume the role.

Important

AWS recommends using IAM role principals in your policies instead of role session principals wherever possible. Use `Condition` statements and condition keys to further scope down access when required.

To specify the role session principal ARN in the `Principal` element, use the following format:

```
"Principal": { "AWS": "arn:aws:sts::AWS-account-ID:assumed-role/role-name/role-session-name" }
```

Additionally, administrators can design a process to control how role sessions are issued. For example, they can provide a one-click solution for their users that creates a predictable session name. If your administrator does this, you can use role session principals in your policies or condition keys. Otherwise, you can specify the role ARN as a principal in the `aws:PrincipalArn` condition key. How you specify the role as a principal can change the effective permissions for the resulting session. For more information, see IAM role principals.

OIDC federated principals

An OIDC federated principal is the principal used when calling AWS STS `AssumeRoleWithWebIdentity` API with a JSON web token (JWT) from an OIDC compliant IDP, also known as OpenID Provider (OP), to request temporary AWS credentials. An OIDC federated principal can represent an OIDC IDP in your AWS account, or the 4 built in identity providers: Login with Amazon, Google, Facebook, and Amazon Cognito.

Users, workloads, or systems who've been issued a JWT from their OIDC IDP can call `AssumeRoleWithWebIdentity` using the JWT to request temporary AWS security credentials for an IAM role that is configured to trust the OIDC IDP that issued the JWT. The JWT can be an id token, access token, or a JWT token delivered by any other method as long as it meets the requirements listed by AWS STS. For more information, see Common scenarios and Requesting credentials through an OIDC provider.

Use this principal type in your role trust policy to allow or deny permissions to call `AssumeRoleWIthWebIdentity` using an OIDC IDP that exists in your AWS account, or one of the four built in IDPs. To specify the OIDC federated principal ARN in the `Principal` element of a role trust policy, use one of the following four formats for the built in OIDC IDPs:

```
"Principal": { "Federated": "cognito-identity.amazonaws.com" }
```

```
"Principal": { "Federated": "www.amazon.com" }
```

```
"Principal": { "Federated": "graph.facebook.com" }
```

```
"Principal": { "Federated": "accounts.google.com" }
```

When using a OIDC provider you add to your account, such as GitHub, you specify the provider's ARN in your role's trust policy. This configuration allows you to write IAM policies that control access specifically for users authenticated through your custom identity provider.

```
"Principal": { "Federated": "arn:aws:iam::AWS-account-ID:oidc-provider/full-OIDC-identity-provider-URL" }
```

For example, if GitHub was the trusted web identity provider, the OIDC role session ARN in the `Principal` element of a role trust policy uses the following format:

```
"Principal": { "Federated": "arn:aws:iam::AWS-account-ID:oidc-provider/
tokens.actions.githubusercontent.com" }
```

See Configuring OpenID Connect in Amazon Web Services for more information.

OIDC federated principals are not supported in policy types other than role trust policies.

SAML federated principals

A *SAML federated principal* is a principal that is used when calling AWS STS `AssumeRoleWithSAML` API to request temporary AWS credentials using a SAML assertion. You can use your SAML identity provider (IDP) to sign in, and then assume an IAM role using this operation. Similar to `AssumeRoleWithWebIdentity`, `AssumeRoleWithSAML` doesn't require AWS credentials for authentication. Instead, users first authenticate with their SAML identity provider, then make the `AssumeRoleWithSAML` API call using their SAML assertion, or are redirected to the AWS Sign-In/SAML page to sign into the AWS Management Console. For more information about which principals can assume a role using this operation, see Compare AWS STS credentials.

Use this principal type in your role trust policy to allow or deny permissions based on the trusted SAML identity provider. To specify the SAML identity role session ARN in the `Principal` element of a role trust policy, use the following format:

```
"Principal": { "Federated": "arn:aws:iam::AWS-account-ID:saml-provider/provider-name" }
```

IAM user principals

You can specify IAM users in the `Principal` element of a resource-based policy or in condition keys that support principals.

> **Note**
>
> In a `Principal` element, the user name part of the *Amazon Resource Name* (ARN) is case sensitive.

```
"Principal": { "AWS": "arn:aws:iam::AWS-account-ID:user/user-name" }
```

```
"Principal": {
  "AWS": [
    "arn:aws:iam::AWS-account-ID:user/user-name-1",
    "arn:aws:iam::AWS-account-ID:user/user-name-2"
  ]
}
```

When you specify users in a `Principal` element, you cannot use a wildcard (*) to mean "all users". Principals must always name specific users.

Important

If your `Principal` element in a role trust policy contains an ARN that points to a specific IAM user, then IAM transforms the ARN to the user's unique principal ID when you save the policy. This helps mitigate the risk of someone escalating their privileges by removing and recreating the user. You don't normally see this ID in the console, because there is also a reverse transformation back to the user's ARN when the trust policy is displayed. However, if you delete the user, then you break the relationship. The policy no longer applies, even if you recreate the user. That's because the new user has a new principal ID that does not match the ID stored in the trust policy. When this happens, the principal ID appears in resource-based policies because AWS can no longer map it back to a valid ARN. The result is that if you delete and recreate a user referenced in a trust policy `Principal` element, you must edit the role to replace the now incorrect principal ID with the correct ARN. IAM once again transforms ARN into the user's new principal ID when you save the policy.

IAM Identity Center principals

In IAM Identity Center, the principal in a resource-based policy must be defined as the AWS account principal. To specify access, reference the role ARN of the permission set in the condition block. For details, see Referencing permission sets in resource policies, Amazon EKS, and AWS KMS in the *IAM Identity Center User Guide*.

AWS STS federated user principals

You can specify *federated user sessions* in the `Principal` element of a resource-based policy or in condition keys that support principals.

> **Important**
>
> AWS recommends that you limit the use of AWS STS federated user sessions. Instead, use IAM roles.

An AWS STS federated user principal is created through the `GetFederationToken` operation called with long-lived IAM credentials. Federated user permissions are the intersection of the principal that called `GetFederationToken` and the session policies passed as parameters to the `GetFederationToken` API.

In AWS, IAM users or an AWS account root user can authenticate using long-term access keys. For more information about which principals can federate using this operation, see Compare AWS STS credentials.

- **IAM federated user** – An IAM user federates using the `GetFederationToken` operation that results in a federated user session for that IAM user.
- **Federated root user** – A root user federates using the `GetFederationToken` operation that results in a federated user session for that root user.

When an IAM user or root user requests temporary credentials from AWS STS using this operation, they begin a temporary federated user session. This session's ARN is based on the original identity that was federated.

To specify the federated user session ARN in the `Principal` element, use the following format:

```
"Principal": { "AWS": "arn:aws:sts::AWS-account-ID:federated-user/user-name" }
```

AWS service principals

You can specify AWS services in the `Principal` element of a resource-based policy or in condition keys that support principals. A *service principal* is an identifier for a service.

IAM roles that can be assumed by an AWS service are called *service roles*. Service roles must include a trust policy. *Trust policies* are resource-based policies attached to a role that defines which principals can assume the role. Some service roles have predefined trust policies. However, in some cases, you must specify the service principal in the trust policy. The service principal in an IAM policy can't be `"Service": "*"`.

> **Important**
>
> The identifier for a service principal includes the service name, and is usually in the following format:
> *service-name*.amazonaws.com

The service principal is defined by the service. You can find the service principal for some services by opening AWS services that work with IAM, checking whether the service has **Yes** in the **Service-linked role** column, and opening the **Yes** link to view the service-linked role documentation for that service. Find the **Service-Linked Role Permissions** section for that service to view the service principal.

The following example shows a policy that can be attached to a service role. The policy enables two services, Amazon ECS and Elastic Load Balancing, to assume the role. The services can then perform any tasks granted by the permissions policy assigned to the role (not shown). To specify multiple service principals, you do not specify two `Service` elements; you can have only one. Instead, you use an array of multiple service principals as the value of a single `Service` element.

```
"Principal": {
    "Service": [
        "ecs.amazonaws.com",
        "elasticloadbalancing.amazonaws.com"
   ]
}
```

AWS service principals in opt-in Regions

You can launch resources in several AWS Regions and some of those Regions you must opt in to. For a complete list of Regions you must opt in to, see Managing AWS Regions in the *AWS General Reference* guide.

When an AWS service in an opt-in Region makes a request within the same Region, the service principal name format is identified as the non-regionalized version of their service principal name:

service-name.amazonaws.com

When an AWS service in an opt-in Region makes a cross-region request to another Region, the service principal name format is identified as the regionalized version of their service principal name:

service-name.*{region}*.amazonaws.com

For example, you have an Amazon SNS topic located in Region ap-southeast-1 and an Amazon S3 bucket located in opt-in Region ap-east-1. You want to configure S3 bucket notifications to publish messages to the SNS topic. To allow the S3 service to post messages to the SNS topic you must grant the S3 service principal sns:Publish permission through the resource-based access policy of the topic.

If you specify the non-regionalized version of the S3 service principal, s3.amazonaws.com, in the topic access policy, the sns:Publish request from the bucket to the topic will fail. The following example specifies the non-regionalized S3 service principal in the Principal policy element of the SNS topic access policy.

```
"Principal": { "Service": "s3.amazonaws.com" }
```

Since the bucket is located in an opt-in Region and the request is made outside of that same Region, the S3 service principal appears as the regionalized service principal name, s3.ap-east-1.amazonaws.com. You must use the regionalized service principal name when an AWS service in an opt-in Region makes a request to another Region. After you specify the regionalized service principal name, if the bucket makes an sns:Publish request to the SNS topic located in another Region, the request will be successful. The following example specifies the regionalized S3 service principal in the Principal policy element of the SNS topic access policy.

```
"Principal": { "Service": "s3.ap-east-1.amazonaws.com" }
```

Resource policies or service principal-based allow-lists for cross-Region requests from an opt-in Region to another Region will only be successful if you specify the regionalized service principal name.

> **Note**
>
> For IAM role trust policies, we recommend using the non-regionalized service principal name. IAM resources are global and therefore the same role can be used in any Region.

All principals

You can use a wildcard (*) to specify all principals in the `Principal` element of a resource-based policy or in condition keys that support principals. Resource-based policies *grant* permissions and condition keys are used to limit the conditions of a policy statement.

> **Important**
>
> We strongly recommend that you do not use a wildcard (*) in the `Principal` element of a resource-based policy with an `Allow` effect unless you intend to grant public or anonymous access. Otherwise, specify intended principals, services, or AWS accounts in the `Principal` element and then further restrict access in the `Condition` element. This is especially true for IAM role trust policies, because they allow other principals to become a principal in your account.

For resource-based policies, using a wildcard (*) with an `Allow` effect grants access to all users, including anonymous users (public access). For IAM users and role principals within your account, no other permissions are required. For principals in other accounts, they must also have identity-based permissions in their account that allow them to access your resource. This is called cross-account access.

For anonymous users, the following elements are equivalent:

```
"Principal": "*"
```

```
"Principal" : { "AWS" : "*" }
```

You cannot use a wildcard to match part of a principal name or ARN.

The following example shows a resource-based policy that can be used instead of AWS JSON policy elements: NotPrincipal to explicitly deny all principals *except* for the ones specified in the `Condition` element. This policy should be added to an Amazon S3 bucket.

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "UsePrincipalArnInsteadOfNotPrincipalWithDeny",
      "Effect": "Deny",
      "Action": "s3:*",
      "Principal": "*",
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ],
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::444455556666:user/user-name"
        }
      }
    }
  ]
}
```

More information

For more information, see the following:

- Bucket policy examples in the *Amazon Simple Storage Service User Guide*
- Example policies for Amazon SNS in the *Amazon Simple Notification Service Developer Guide*
- Amazon SQS policy examples in the *Amazon Simple Queue Service Developer Guide*
- Key policies in the *AWS Key Management Service Developer Guide*
- Account identifiers in the *AWS General Reference*
- OIDC federation

AWS JSON policy elements: NotPrincipal

The `NotPrincipal` element uses `"Effect":"Deny"` to deny access to all principals ***except*** the principal specified in the `NotPrincipal` element. A principal can be an IAM user, AWS STS federated user principal, IAM role, assumed role session, AWS account, AWS service, or other principal type. For more information about principals, see AWS JSON policy elements: Principal.

`NotPrincipal` must be used with `"Effect":"Deny"`. Using it with `"Effect":"Allow"` is not supported.

Important

We do not recommend the use of `NotPrincipal` for new resource-based policies as part of your security and authorization strategy. When you use `NotPrincipal`, troubleshooting the effects of multiple policy types can be difficult. We recommend using the `aws:PrincipalArn` context key with ARN condition operators instead.

Key points

- The `NotPrincipal` element is supported in resource-based policies for some AWS services, including VPC endpoints. Resource-based policies are policies that you embed directly in a resource. You cannot use the `NotPrincipal` element in an IAM identity-based policy nor in an IAM role trust policy.
- Don't use resource-based policy statements that include a `NotPrincipal` policy element with a `Deny` effect for IAM users or roles that have a permissions boundary policy attached. The `NotPrincipal` element with a `Deny` effect will always deny any IAM principal that has a permissions boundary policy attached, regardless of the values specified in the `NotPrincipal` element. This causes some IAM users or roles that would otherwise have access to the resource to lose access. We recommend changing your resource-based policy statements to use the condition operator `ArnNotEquals` with the `aws:PrincipalArn` context key to limit access instead of the `NotPrincipal` element. For information about permissions boundaries, see Permissions boundaries for IAM entities.
- When you use `NotPrincipal`, you must also specify the account ARN of the not-denied principal. Otherwise, the policy might deny access to the entire account containing the principal. Depending on the service that you include in your policy, AWS might validate the account first and then the user. If an assumed-role user (someone who is using a role) is being evaluated, AWS might validate the account first, then the role, and then the assumed-role user. The assumed-role user is identified by the role session name that is specified when they assumed the role. Therefore, we strongly recommend that you explicitly include the ARN for a user's account, or include both the ARN for a role and the ARN for the account containing that role.
- The `NotPrincipal` element isn't supported in Service Control Policies (SCP) and Resource Control Policies (RCP).

Alternatives to the `NotPrincipal` element

When managing access control in AWS, there may be scenarios where you need to explicitly deny all principals access to a resource, except for one or more principals you specify. AWS recommends using a Deny statement with global condition context keys for more precise control and easier troubleshooting. The following examples show alternative approaches using condition operators like `StringNotEquals` or `ArnNotEquals` to deny access to all principals except those specified in the Condition element.

Example scenario using an IAM role

You can use a resource-based policy with a Deny statement to prevent all IAM roles, except those specified in the Condition element, from accessing or manipulating your resources. This approach follows the AWS security principle that an explicit deny always takes precedence over any allow statements and helps maintain the principle of least privilege across your AWS infrastructure.

Instead of using `NotPrincipal`, we recommend using a Deny statement with global condition context keys and the condition operator like `ArnNotEquals` to explicitly allow an IAM role access to your resources. The following example uses aws:PrincipalArn to explicitly allow the role `read-only-role` to access Amazon S3 buckets in the `Bucket_Account_Audit` folder.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCrossAuditAccess",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::Bucket_Account_Audit",
        "arn:aws:s3:::Bucket_Account_Audit/*"
      ],
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::444455556666:role/read-only-role"
        }
      }
    }
```

```
  ]
}
```

Example scenario using a service principal

You can use a Deny statement to prevent all service principals, except those specified in the `Condition` element, from accessing or manipulating your resources. This approach is particularly useful when you need to implement fine-grained access controls or establish security boundaries between different services and applications in your AWS environment.

Instead of using `NotPrincipal`, we recommend using a Deny statement with global condition context keys and the condition operator `StringNotEquals` to explicitly allow a service principal access to your resources. The following example uses `aws:PrincipalServiceName` to explicitly allow the AWS CodeBuild service principal to access Amazon S3 buckets in the BUCKETNAME folder.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyNotCodeBuildAccess",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::BUCKETNAME",
        "arn:aws:s3:::BUCKETNAME/*"
      ],
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:PrincipalServiceName": "codebuild.amazonaws.com"
        }
      }
    }
  ]
}
```

IAM JSON policy elements: Action

The `Action` element describes the specific action or actions that will be allowed or denied. Statements must include either an `Action` or `NotAction` element. Each AWS service has its own set of actions that describe tasks that you can perform with that service. For example, the list of actions for Amazon S3 can be found at Specifying Permissions in a Policy in the *Amazon Simple Storage Service User Guide*, the list of actions for Amazon EC2 can be found in the Amazon EC2 API Reference, and the list of actions for AWS Identity and Access Management can be found in the IAM API Reference. To find the list of actions for other services, consult the API reference documentation for the service.

AWS also provides service reference information in JSON format to streamline the automation of policy management workflows. With the service reference information, you can access available actions, resources, and condition keys across AWS services from machine-readable files. For more information, see Simplified AWS service information for programmatic access in the Service Authorization Reference.

You specify a value using a service namespace as an action prefix (`iam`, `ec2`, `sqs`, `sns`, `s3`, etc.) followed by the name of the action to allow or deny. The name must match an action that is supported by the service. The prefix and the action name are case insensitive. For example, `iam:ListAccessKeys` is the same as `IAM:listaccesskeys`. The following examples show `Action` elements for different services.

Amazon SQS action

```
"Action": "sqs:SendMessage"
```

Amazon EC2 action

```
"Action": "ec2:StartInstances"
```

IAM action

```
"Action": "iam:ChangePassword"
```

Amazon S3 action

```
"Action": "s3:GetObject"
```

You can specify multiple values for the `Action` element.

```
"Action": [ "sqs:SendMessage", "sqs:ReceiveMessage", "ec2:StartInstances",
 "iam:ChangePassword", "s3:GetObject" ]
```

You can use multi-character match wildcards (*) and single-character match wildcards (?) to give access to all the actions the specific AWS product offers. For example, the following `Action` element applies to all S3 actions.

```
"Action": "s3:*"
```

You can also use wildcards (* or ?) as part of the action name. For example, the following `Action` element applies to all IAM actions that include the string `AccessKey`, including `CreateAccessKey`, `DeleteAccessKey`, `ListAccessKeys`, and `UpdateAccessKey`.

```
"Action": "iam:*AccessKey*"
```

Some services let you limit the actions that are available. For example, Amazon SQS lets you make available just a subset of all the possible Amazon SQS actions. In that case, the * wildcard doesn't allow complete control of the queue; it allows only the subset of actions that you've shared. For more information, see Understanding Permissions in the *Amazon Simple Queue Service Developer Guide*.

IAM JSON policy elements: NotAction

`NotAction` is an advanced policy element that explicitly matches everything *except* the specified list of actions. Using `NotAction` can result in a shorter policy by listing only a few actions that should not match, rather than including a long list of actions that will match. Actions specified in `NotAction` are not impacted by the `Allow` or `Deny` effect in a policy statement. This, in turn, means that all of the applicable actions or services that are not listed are allowed if you use the `Allow` effect. In addition, such unlisted actions or services are denied if you use the `Deny` effect. When you use `NotAction` with the `Resource` element, you provide scope for the policy. This is how AWS determines which actions or services are applicable. For more information, see the following example policy.

NotAction with Allow

You can use the `NotAction` element in a statement with `"Effect": "Allow"` to provide access to all of the actions in an AWS service, except for the actions specified in `NotAction`. You can use

it with the `Resource` element to provide scope for the policy, limiting the allowed actions to the actions that can be performed on the specified resource.

The following example allows users to access all of the Amazon S3 actions that can be performed on any S3 resource *except* for deleting a bucket. This policy also does not allow actions in other services, because other service actions are not applicable to the S3 resources.

```
"Effect": "Allow",
"NotAction": "s3:DeleteBucket",
"Resource": "arn:aws:s3:::*",
```

Sometimes, you might want to allow access to a large number of actions. By using the `NotAction` element you effectively reverse the statement, resulting in a shorter list of actions. For example, because AWS has so many services, you might want to create a policy that allows the user to do everything except access IAM actions.

The following example allows users to access every action in every AWS service except for IAM.

```
"Effect": "Allow",
"NotAction": "iam:*",
"Resource": "*"
```

Be careful using the `NotAction` element and `"Effect": "Allow"` in the same statement or in a different statement within a policy. `NotAction` matches all services and actions that are not explicitly listed or applicable to the specified resource, and could result in granting users more permissions than you intended.

NotAction with Deny

You can use the `NotAction` element in a statement with `"Effect": "Deny"` to deny access to all of the listed resources except for the actions specified in the `NotAction` element. This combination does not allow the listed items, but instead explicitly denies the actions not listed. You must still allow actions that you want to allow.

The following conditional example denies access to non-IAM actions if the user is not signed in using MFA. If the user is signed in with MFA, then the `"Condition"` test fails and the final `"Deny"` statement has no effect. Note, however, that this would not grant the user access to any actions; it would only explicitly deny all other actions except IAM actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Sid": "DenyAllUsersNotUsingMFA",
        "Effect": "Deny",
        "NotAction": "iam:*",
        "Resource": "*",
        "Condition": {"BoolIfExists": {"aws:MultiFactorAuthPresent": "false"}}
    }]
}
```

For an example policy that denies access to actions outside of specific Regions, except for actions from specific services, see AWS: Denies access to AWS based on the requested Region.

IAM JSON policy elements: Resource

The `Resource` element in an IAM policy statement defines the object or objects that the statement applies to. Statements must include either a `Resource` or a `NotResource` element.

You specify a resource using an Amazon Resource Name (ARN). The format of the ARN depends on the AWS service and the specific resource you're referring to. Although the ARN format varies you always use an ARN to identify a resource. For more information about the format of ARNs, see IAM ARNs. For information about how to specify a resource, refer to the documentation for the service you want to write a statement.

> **Note**
>
> Some AWS services do not allow you to specify actions for individual resources. In these cases, any actions that you list in the `Action` or `NotAction` element apply to all resources in that service. When this is the case, you use the wildcard character (`*`) in the `Resource` element.

The following example refers to a specific Amazon SQS queue.

```
"Resource": "arn:aws:sqs:us-east-2:account-ID-without-hyphens:queue1"
```

The following example refers to the IAM user named Bob in an AWS account.

> **Note**
>
> In the `Resource` element, the IAM user name is case sensitive.

```
"Resource": "arn:aws:iam::account-ID-without-hyphens:user/Bob"
```

Using wildcards in resource ARNs

You can use wildcard characters (* and ?) within the individual segments of an ARN (the parts separated by colons) to represent:

- Any combination of characters (*)
- Any single character (?)

You can use multiple * or ? characters in each segment. If the * wildcard is the last character of a resource ARN segment, it can expand to match beyond the colon boundaries. We recommend you use wildcards (* and ?) within ARN segments separated by a colon.

> **Note**
>
> You can't use a wildcard in the service segment that identifies the AWS product. For more information about ARN segments, see Identify AWS resources with Amazon Resource Names (ARNs)

The following example refers to all IAM users whose path is `/accounting`.

```
"Resource": "arn:aws:iam::account-ID-without-hyphens:user/accounting/*"
```

The following example refers to all items within a specific Amazon S3 bucket.

```
"Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
```

The asterisk (*) character can expand to replace everything within a segment, including characters like a forward slash (/) that may otherwise appear to be a delimiter within a given service

namespace. For example, consider the following Amazon S3 ARN as the same wildcard expansion logic applies to all services.

```
"Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*/test/*"
```

The wildcards in the ARN apply to all of the following objects in the bucket, not only the first object listed.

```
amzn-s3-demo-bucket/1/test/object.jpg
amzn-s3-demo-bucket/1/2/test/object.jpg
amzn-s3-demo-bucket/1/2/test/3/object.jpg
amzn-s3-demo-bucket/1/2/3/test/4/object.jpg
amzn-s3-demo-bucket/1///test///object.jpg
amzn-s3-demo-bucket/1/test/.jpg
amzn-s3-demo-bucket//test/object.jpg
amzn-s3-demo-bucket/1/test/
```

Consider the last two objects in the previous list. An Amazon S3 object name can begin or end with the conventional delimiter forward slash (/) character. While / works as a delimiter, there is no specific significance when this character is used within a resource ARN. It is treated the same as any other valid character. The ARN would not match the following objects:

```
amzn-s3-demo-bucket/1-test/object.jpg
amzn-s3-demo-bucket/test/object.jpg
amzn-s3-demo-bucket/1/2/test.jpg
```

Specifying multiple resources

You can specify multiple resources in the `Resource` element by using an array of ARNs. The following example refers to two DynamoDB tables.

```
"Resource": [
    "arn:aws:dynamodb:us-east-2:account-ID-without-hyphens:table/books_table",
    "arn:aws:dynamodb:us-east-2:account-ID-without-hyphens:table/magazines_table"
]
```

Using policy variables in resource ARNs

In the `Resource` element, you can use JSON policy variables in the part of the ARN that identifies the specific resource (that is, in the trailing part of the ARN). For example, you can use the key

{aws:username} as part of a resource ARN to indicate that the current user's name should be included as part of the resource's name. The following example shows how you can use the {aws:username} key in a Resource element. The policy allows access to a Amazon DynamoDB table that matches the current user's name.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "dynamodb:*",
        "Resource": "arn:aws:dynamodb:us-east-2:111122223333:table/
${aws:username}"
    }
}
```

For more information about JSON policy variables, see IAM policy elements: Variables and tags.

IAM JSON policy elements: NotResource

NotResource is an advanced policy element that explicitly matches every resource except those specified. Using NotResource can result in a shorter policy by listing only a few resources that should not match, rather than including a long list of resources that will match. This is particularly useful for policies that apply within a single AWS service.

For example, imagine you have a group named HRPayroll. Members of HRPayroll should not be allowed to access any Amazon S3 resources except the Payroll folder in the HRBucket bucket. The following policy explicitly denies access to all Amazon S3 resources other than the listed resources. Note, however, that this policy does not grant the user access to any resources.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "s3:*",
    "NotResource": [
```

```
      "arn:aws:s3:::HRBucket/Payroll",
      "arn:aws:s3:::HRBucket/Payroll/*"
    ]
  }
}
```

Normally, to explicitly deny access to a resource you would write a policy that uses `"Effect":"Deny"` and that includes a `Resource` element that lists each folder individually. However, in that case, each time you add a folder to `HRBucket`, or add a resource to Amazon S3 that should not be accessed, you must add its name to the list in the `Resource` element. If you use a `NotResource` element instead, users are automatically denied access to new folders unless you add the folder names to the `NotResource` element.

When using `NotResource`, you should keep in mind that resources specified in this element are the *only* resources that are not limited. This, in turn, limits all of the resources that would apply to the action. In the example above, the policy affects only Amazon S3 actions, and therefore only Amazon S3 resources. If the `Action` element also included Amazon EC2 actions, then the policy would deny access to any EC2 resources not specified in the `NotResource` element. To learn which actions in a service allow specifying the ARN of a resource, see Actions, Resources, and Condition Keys for AWS Services.

NotResource with other elements

You should **never** use the `"Effect": "Allow"`, `"Action": "*"`, and `"NotResource": "arn:aws:s3:::HRBucket"` elements together. This statement is very dangerous, because it allows all actions in AWS on all resources except the `HRBucket` S3 bucket. This would even allow the user to add a policy to themselves that allows them to access `HRBucket`. Do not do this.

Be careful using the `NotResource` element and `"Effect": "Allow"` in the same statement or in a different statement within a policy. `NotResource` allows all services and resources that are not explicitly listed, and could result in granting users more permissions than you intended. Using the `NotResource` element and `"Effect": "Deny"` in the same statement denies services and resources that are not explicitly listed.

IAM JSON policy elements: Condition

The `Condition` element (or `Condition` *block*) lets you specify conditions for when a policy is in effect. The `Condition` element is optional. In the `Condition` element, you build expressions in

which you use condition operators (equal, less than, and others) to match the context keys and values in the policy against keys and values in the request context. To learn more about the request context, see Components of a request.

```
"Condition" : { "{condition-operator}" : { "{condition-key}" : "{condition-value}" }}
```

The context key that you specify in a policy condition can be a global condition context key or a service-specific context key. Global condition context keys have the `aws:` prefix. Service-specific context keys have the service's prefix. For example, Amazon EC2 lets you write a condition using the `ec2:InstanceType` context key, which is unique to that service. To view service-specific IAM context keys with the `iam:` prefix, see IAM and AWS STS condition context keys.

Context key *names* are not case-sensitive. For example, including the `aws:SourceIP` context key is equivalent to testing for `AWS:SourceIp`. Case-sensitivity of context key *values* depends on the condition operator that you use. For example, the following condition includes the `StringEquals` operator to make sure that only requests made by `john` match. Users named `John` are denied access.

```
"Condition" : { "StringEquals" : { "aws:username" : "john" }}
```

The following condition uses the `StringEqualsIgnoreCase` operator to match users named `john` or `John`.

```
"Condition" : { "StringEqualsIgnoreCase" : { "aws:username" : "john" }}
```

Some context keys support key–value pairs that allow you to specify part of the key name. Examples include the `aws:RequestTag/tag-key` context key, the AWS KMS `kms:EncryptionContext:encryption_context_key`, and the `ResourceTag/tag-key` context key supported by multiple services.

- If you use the `ResourceTag/tag-key` context key for a service such as Amazon EC2, then you must specify a key name for the `tag-key`.
- **Key names are not case-sensitive.** This means that if you specify `"aws:ResourceTag/TagKey1": "Value1"` in the condition element of your policy, then the condition matches a resource tag key named either `TagKey1` or `tagkey1`, but not both.
- AWS services that support these attributes might allow you to create multiple key names that differ only by case. For example, you might tag an Amazon EC2 instance with `ec2=test1` and

`EC2=test2`. When you use a condition such as `"aws:ResourceTag/EC2": "test1"` to allow access to that resource, the key name matches both tags, but only one value matches. This can result in unexpected condition failures.

> **Important**
>
> As a best practice, make sure that members of your account follow a consistent naming convention when naming key–value pair attributes. Examples include tags or AWS KMS encryption contexts. You can enforce this using the `aws:TagKeys` context key for tagging, or the `kms:EncryptionContextKeys` for the AWS KMS encryption context.

- For a list of all of the condition operators and a description of how they work, see Condition operators.
- Unless otherwise specified, all context keys can have multiple values. For a description of how to handle context keys that have multiple values, see Set operators for multivalued context keys.
- For a list of all of the globally available context keys, see AWS global condition context keys.
- For condition context keys that are defined by each service, see Actions, Resources, and Condition Keys for AWS Services.

The request context

When a principal makes a request to AWS, AWS gathers the request information into a request context. The request context includes information about the principal, resources, actions, and other environmental properties. Policy evaluation matches the properties in the policy against the properties sent in the request to evaluate and authorize actions you can perform in AWS.

You can use the `Condition` element of a JSON policy to test specific context keys against the request context. For example, you can create a policy that uses the aws:CurrentTime context key to allow a user to perform actions within only a specific range of dates.

The following example shows a representation of the request context when Martha Rivera sends a request to deactivate her MFA device.

```
Principal: AROA123456789EXAMPLE
Action: iam:DeactivateMFADevice
Resource: arn:aws:iam::user/martha
```

```
Context:
  - aws:UserId=AROA123456789EXAMPLE:martha
  - aws:PrincipalAccount=1123456789012
  - aws:PrincipalOrgId=o-example
  - aws:PrincipalARN=arn:aws:iam::1123456789012:assumed-role/TestAR
  - aws:MultiFactorAuthPresent=true
  - aws:MultiFactorAuthAge=2800
  - aws:CurrentTime=...
  - aws:EpochTime=...
  - aws:SourceIp=...
```

The request context is matched against a policy that allows users to remove their own multi-factor authentication (MFA) device, but only if they have signed in using MFA in the last hour (3,600 seconds).

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "AllowRemoveMfaOnlyIfRecentMfa",
        "Effect": "Allow",
        "Action": [
            "iam:DeactivateMFADevice"
        ],
        "Resource": "arn:aws:iam::*:user/${aws:username}",
        "Condition": {
            "NumericLessThanEquals": {"aws:MultiFactorAuthAge": "3600"}
        }
    }
}
```

In this example, the policy matches the request context: the action is the same, the resource matches the "*" wildcard, and the value for `aws:MultiFactorAuthAge` is 2800, which is less than 3600, so the policy allows this authorization request.

AWS evaluates each context key in the policy and returns a value of *true* or *false*. A context key that is not present in the request is considered a mismatch.

The request context can return the following values:

- **True** – If the requester signed in using MFA in the last one hour or less, then the condition returns *true*.
- **False** – If the requester signed in using MFA more than one hour ago, then the condition returns *false*.
  - **Not present** – If the requester made a request using their IAM user access keys in the AWS CLI or AWS API, the key is not present. In this case, the key is not present, and it won't match.

> **Note**
>
> In some cases, when the condition key value is not present, the condition can still return true. For example, if you add the `ForAllValues` qualifier, the request returns true if the context key is not in the request. To prevent missing context keys or context keys with empty values from evaluating to true, you can include the Null condition operator in your policy with a `false` value to check if the context key exists and its value is not null.

The condition block

The following example shows the basic format of a `Condition` element:

```
"Condition": {"StringLike": {"s3:prefix": ["jane/*"]}}
```

A value from the request is represented by a context key, in this case `s3:prefix`. The context key value is compared to a value that you specify as a literal value, such as `jane/*`. The type of comparison to make is specified by the condition operator (here, `StringLike`). You can create conditions that compare strings, dates, numbers, and more using typical Boolean comparisons such as equals, greater than, and less than. When you use string operators or ARN operators, you can also use a policy variable in the context key value. The following example includes the `aws:username` variable.

```
"Condition": {"StringLike": {"s3:prefix": ["${aws:username}/*"]}}
```

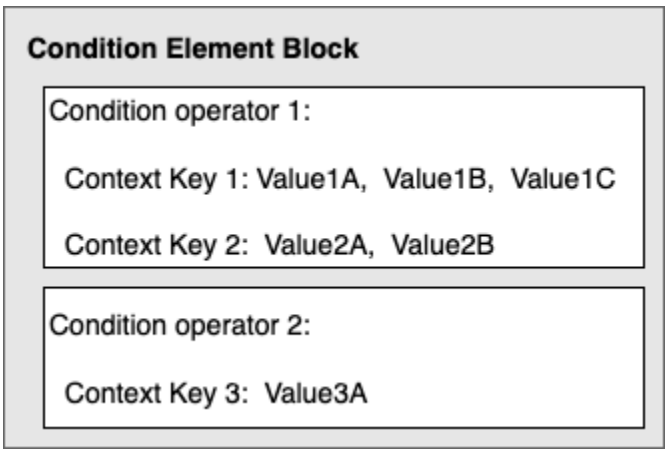
Under some circumstances, context keys can contain multiple values. For example, a request to Amazon DynamoDB might ask to return or update multiple attributes from a table. A policy for access to DynamoDB tables can include the `dynamodb:Attributes` context key, which contains all the attributes listed in the request. You can test the multiple attributes in the request against

a list of allowed attributes in a policy by using set operators in the `Condition` element. For more information, see Set operators for multivalued context keys.

When the policy is evaluated during a request, AWS replaces the key with the corresponding value from the request. (In this example, AWS would use the date and time of the request.) The condition is evaluated to return true or false, which is then factored into whether the policy as a whole allows or denies the request.

Multiple values in a condition

A `Condition` element can contain multiple condition operators, and each condition operator can contain multiple context key-value pairs. The following figure illustrates this.



For more information, see Set operators for multivalued context keys.

IAM JSON policy elements: Condition operators

Use condition operators in the `Condition` element to match the condition key and value in the policy against values in the request context. For more information about the `Condition` element, see IAM JSON policy elements: Condition.

The condition operator that you can use in a policy depends on the condition key you choose. You can choose a global condition key or a service-specific condition key. To learn which condition operator you can use for a global condition key, see AWS global condition context keys. To learn which condition operator you can use for a service-specific condition key, see Actions, Resources, and Condition Keys for AWS Services and choose the service that you want to view.

Important

If the key that you specify in a policy condition is not present in the request context, the values do not match and the condition is *false*. If the policy condition requires that the key is *not* matched, such as `StringNotLike` or `ArnNotLike`, and the right key is not present, the condition is *true*. This logic applies to all condition operators except ...IfExists and Null check. These operators test whether the key is present (exists) in the request context.

The condition operators can be grouped into the following categories:

- String
- Numeric
- Date and time
- Boolean
- Binary
- IP address
- Amazon Resource Name (ARN) (available for only some services.)
- ...IfExists (checks if the key value exists as part of another check)
- Null check (checks if the key value exists as a standalone check)

String condition operators

String condition operators let you construct `Condition` elements that restrict access based on comparing a key to a string value.

- **Policy variables** – Supported
- **Wildcards** – Supported

Condition operator	Description
`StringEquals`	Exact matching, case sensitive
`StringNotEquals`	Negated matching

Condition operator	Description
`StringEqualsIgnoreCase`	Exact matching, ignoring case
`StringNotEqualsIgnoreCase`	Negated matching, ignoring case
`StringLike`	Case-sensitive matching. The values can include multi-character match wildcards (*) and single-character match wildcards (?) anywhere in the string. You must specify wildcards to achieve partial string matches. **Note** If a key contains multiple values, `StringLike` can be qualified with set operators—`ForAllValues:StringLike` and `ForAnyValue:StringLike`. For more information, see Set operators for multivalued context keys.
`StringNotLike`	Negated case-sensitive matching. The values can include multi-character match wildcards (*) or single-character match wildcards (?) anywhere in the string.

Example string condition operator

For example, the following statement contains a `Condition` element that uses `aws:PrincipalTag` key to specify that the principal making the request must be tagged with the `iamuser-admin` job category.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
```

```
        "Action": "iam:*AccessKey*",
        "Resource": "arn:aws:iam::111122223333:user/*",
        "Condition": {
            "StringEquals": {
                "aws:PrincipalTag/job-category": "iamuser-admin"
            }
        }
    }
}
```

If the key that you specify in a policy condition is not present in the request context, the values do not match. In this example, the `aws:PrincipalTag/job-category` key is present in the request context if the principal is using an IAM user with attached tags. It is also included for a principal using an IAM role with attached tags or session tags. If a user without the tag attempts to view or edit an access key, the condition returns `false` and the request is implicitly denied by this statement.

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy Condition	Request Context	Result
`"StringEquals": { "aws:PrincipalTag/job-category": "iamuser-admin" }`	`aws:PrincipalTag/job-category: - iamuser-admin`	Match
`"StringEquals": { "aws:PrincipalTag/job-category": "iamuser-admin" }`	`aws:PrincipalTag/job-category: - dev-ops`	No match
`"StringEquals": { "aws:PrincipalTag/job-category": "iamuser-admin"`	No `aws:PrincipalTag/job-category` in the request context.	No match

Policy Condition	Request Context	Result
}		

Example using a policy variable with a string condition operator

The following example uses the `StringLike` condition operator to perform string matching with a policy variable to create a policy that lets an IAM user use the Amazon S3 console to manage his or her own "home directory" in an Amazon S3 bucket. The policy allows the specified actions on an S3 bucket as long as the `s3:prefix` matches any one of the specified patterns.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Condition": {
        "StringLike": {
          "s3:prefix": [
            "",
            "home/",
            "home/${aws:username}/"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
```

```
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:username}",
        "arn:aws:s3:::amzn-s3-demo-bucket/home/${aws:username}/*"
      ]
    }
  ]
}
```

The following table shows how AWS evaluates this policy for different users based on the aws:username value in the request context.

Policy condition	Request context	Result
`"StringLike": { "s3:prefix": [ "home/", "home/${aws:username}/" ] }`	`aws:username: - martha_rivera`	`"StringLike": { "s3:prefix": [ "home/", "home/martha_rivera /" ] }`
`"StringLike": { "s3:prefix": [ "home/", "home/${aws:username}/" ] }`	`aws:username: - nikki_wolf`	`"StringLike": { "s3:prefix": [ "home/", "home/nikki_wolf /" ] }`
`"StringLike": { "s3:prefix": [ "home/", "home/${aws:username}/" ] }`	No `aws:username` in the request context.	No match

For an example of a policy that shows how to use the `Condition` element to restrict access to resources based on an application ID and a user ID for OIDC federation, see Amazon S3: Allows Amazon Cognito users to access objects in their bucket.

Multivalued string condition operators

If a key in the request contains multiple values, string operators can be qualified with set operators `ForAllValues` and `ForAnyValue`. For more information on the evaluation logic of multiple context keys or values, see Set operators for multivalued context keys.

Condition operator	Description

`ForAllValues:StringEquals`
 	
`ForAllValues:StringEqualsIgnoreCase`
	All of the values for the condition key in the request must match at least one of the values in your policy.

`ForAnyValue:StringEquals`
 	
`ForAnyValue:StringEqualsIgnoreCase`
	At least one condition key value in the request must match one of the values in your policy.

`ForAllValues:StringNotEquals`
 	
`ForAllValues:StringNotEqualsIgnoreCase`
	Negated matching.
	
	None of the values of the context key in the request can match any of the context key values in your policy.

`ForAnyValue:StringNotEquals`
 	
`ForAnyValue:StringNotEqualsIgnoreCase`
	Negated matching.
	
	At least one context key value in the request must NOT match any of values in the context key in your policy.

`ForAllValues:StringLike`	All of the values for the condition key in the request must match at least one of the values in your policy.
`ForAnyValue:StringLike`	At least one condition key value in the request must match one of the values in your policy.

Condition operator	Description
`ForAllValues:StringNotLike`	Negated matching. None of the values of the context key in the request can match any of the context key values in your policy.
`ForAnyValue:StringNotLike`	Negated matching. At least one context key value in the request must NOT match any of values in the context key in your policy.

Example using `ForAnyValue` with a string condition operator

This example shows how you might create an identity-based policy that allows using the Amazon EC2 `CreateTags` action to attach tags to an instance. When you use `StringEqualsIgnoreCase`, you can attach tags only if the tag contains the `environment` key with the `preprod` or `storage` values. When you append `IgnoreCase` to the operator, you allow any existing tag value capitalization, such as `preprod`, `Preprod`, and `PreProd`, to resolve to true.

When you add the `ForAnyValue` modifier with the aws:TagKeys condition key, at least one tag key value in the request must match the value `environment`. `ForAnyValue` comparison is case sensitive, which stops users from using the incorrect case for the tag key, such as using `Environment` instead of `environment`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:CreateTags",
    "Resource": "arn:aws:ec2:*:*:instance/*",
    "Condition": {
      "StringEqualsIgnoreCase": {
        "aws:RequestTag/environment": [
          "preprod",
```

```
          "storage"
        ]
      },
      "ForAnyValue:StringEquals": {
        "aws:TagKeys": "environment"
      }
    }
  }
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"StringEqualsIgnoreCase": { "aws:RequestTag/environment": [ "preprod", "storage" ] }, "ForAnyValue:StringEquals": { "aws:TagKeys": "environment" }`	`aws:TagKeys: - environment aws:RequestTag/environment: - preprod`	Match
`"StringEqualsIgnoreCase": { "aws:RequestTag/environment": [ "preprod", "storage" ] }, "ForAnyValue:StringEquals": {`	`aws:TagKeys: - environment - costcenter aws:RequestTag/environment: - PreProd`	Match

Policy condition	Request context	Result
`"aws:TagKeys": "environment" }`		
`"StringEqualsIgnoreCase": { "aws:RequestTag/environment": [ "preprod", "storage" ] }, "ForAnyValue:StringEquals": { "aws:TagKeys": "environment" }`	`aws:TagKeys: - Environment aws:RequestTag/Environment: - preprod`	No match
`"StringEqualsIgnoreCase": { "aws:RequestTag/environment": [ "preprod", "storage" ] }, "ForAnyValue:StringEquals": { "aws:TagKeys": "environment" }`	`aws:TagKeys: - costcenter aws:RequestTag/environment: - preprod`	No match

Policy condition	Request context	Result
`"StringEqualsIgnoreCase": { "aws:RequestTag/environment": [ "preprod", "storage" ] }, "ForAnyValue:StringEquals": { "aws:TagKeys": "environment" }`	No `aws:TagKeys` in the request context. `aws:RequestTag/environment: - storage`	No match
`"StringEqualsIgnoreCase": { "aws:RequestTag/environment": [ "preprod", "storage" ] }, "ForAnyValue:StringEquals": { "aws:TagKeys": "environment" }`	`aws:TagKeys: - environment` No `aws:RequestTag/environment` in the request context.	No match

Policy condition	Request context	Result
`"StringEqualsIgnoreCase": { "aws:RequestTag/environment": [ "preprod", "storage" ] }, "ForAnyValue:StringEquals": { "aws:TagKeys": "environment" }`	No `aws:TagKeys` in the request context. No `aws:RequestTag/environment` in the request context.	No match

Wildcard matching

String condition operators perform a patternless matching that does not enforce a predefined format. ARN and Date condition operators are a subset of string operators that enforce a structure on the condition key value.

We recommend you use condition operators that correspond to the values you're comparing keys to. For example, you should use the section called “String condition operators” when comparing keys to string values. Similarly, you should use the section called “Amazon Resource Name (ARN) condition operators” when comparing keys to ARN values.

Example

This example shows how you might create a boundary around resources in your organization. The condition in this policy denies access to Amazon S3 actions unless the resource being accessed is in a specific set of organizational units (OUs) in AWS Organizations. An AWS Organizations path is a text representation of the structure of an organization's entity.

The condition requires that `aws:ResourceOrgPaths` contains any of the listed OU paths. Because `aws:ResourceOrgPaths` is a multi-value condition, the policy uses the `ForAllValues:StringNotLike` operator to compare the values of `aws:ResourceOrgPaths` to the list of OUs in the policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyS3AccessOutsideMyBoundary",
      "Effect": "Deny",
      "Action": [
        "s3:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringNotLike": {
          "aws:ResourceOrgPaths": [
            "o-acorg/r-acroot/ou-acroot-mediaou/",
            "o-acorg/r-acroot/ou-acroot-sportsou/*"
          ]
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"ForAllValues:StringNotLike": { "aws:ResourceOrgPaths": [ "o-acorg/r-acroot/ou-acroot-mediaou/", "o-acorg/r-acroot/ou-acroot-sportsou/*" ] }`	`aws:ResourceOrgPaths: - o-acorg/r-acroot/ou-acroot-sportsou/costcenter/`	Match

Policy condition	Request context	Result
`"ForAllValues:StringNotLike": { "aws:ResourceOrgPaths": [ "o-acorg/r-acroot/ou-acroot-mediaou/", "o-acorg/r-acroot/ou-acroot-sportsou/*" ] }`	`aws:ResourceOrgPaths: - o-acorg/r-acroot/ou-acroot-mediaou/costcenter/`	No match
`"ForAllValues:StringNotLike": { "aws:ResourceOrgPaths": [ "o-acorg/r-acroot/ou-acroot-mediaou/", "o-acorg/r-acroot/ou-acroot-sportsou/*" ] }`	No `aws:ResourceOrgPaths:` in the request.	No match

Numeric condition operators

Numeric condition operators let you construct `Condition` elements that restrict access based on comparing a key to an integer or decimal value.

- **Policy variables** – Not supported
- **Wildcards** – Not supported

Condition operator	Description
`NumericEquals`	Matching
`NumericNotEquals`	Negated matching

Condition operator	Description
NumericLessThan	"Less than" matching
NumericLessThanEqu als	"Less than or equals" matching
NumericGreaterThan	"Greater than" matching
NumericGreaterThan Equals	"Greater than or equals" matching

For example, the following statement contains a Condition element that uses the NumericLessThanEquals condition operator with the s3:max-keys key to specify that the requester can list *up to* 10 objects in amzn-s3-demo-bucket at a time.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
    "Condition": {"NumericLessThanEquals": {"s3:max-keys": "10"}}
  }
}
```

If the key that you specify in a policy condition is not present in the request context, the values do not match. In this example, the s3:max-keys key is always present in the request when you perform the ListBucket operation. If this policy allowed all Amazon S3 operations, then only the operations that include the max-keys context key with a value of less than or equal to 10 would be allowed.

Date condition operators

Date condition operators let you construct Condition elements that restrict access based on comparing a key to a date/time value. You use these condition operators with aws:CurrentTime

key or aws:EpochTime key. You must specify date/time values with one of the W3C implementations of the ISO 8601 date formats or in epoch (UNIX) time.

- **Policy variables** – Not supported
- **Wildcards** – Not supported

Condition operator	Description
DateEquals	Matching a specific date
DateNotEquals	Negated matching
DateLessThan	Matching before a specific date and time
DateLessThanEquals	Matching at or before a specific date and time
DateGreaterThan	Matching after a specific a date and time
DateGreaterThanEquals	Matching at or after a specific date and time

For example, the following statement contains a Condition element that uses the DateGreaterThan condition operator with the aws:TokenIssueTime key. This condition specifies that the temporary security credentials used to make the request were issued in 2020. This policy can be updated programmatically every day to ensure that account members use fresh credentials.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "iam:*AccessKey*",
        "Resource": "arn:aws:iam::111122223333:user/*",
        "Condition": {
            "DateGreaterThan": {
                "aws:TokenIssueTime": "2020-01-01T00:00:01Z"
```

```
            }
        }
    }
}
```

If the key that you specify in a policy condition is not present in the request context, the values do not match. The `aws:TokenIssueTime` key is present in the request context only when the principal uses temporary credentials to make the request. The key is not present in AWS CLI, AWS API, or AWS SDK requests that are made using access keys. In this example, if an IAM user attempts to view or edit an access key, the request is denied.

Boolean condition operators

Boolean conditions let you construct `Condition` elements that restrict access based on comparing a key to `true` or `false`.

If a key contains multiple values, boolean operators can be qualified with set operators `ForAllValues` and `ForAnyValue`. For more information on the evaluation logic of multiple context keys or values, see Set operators for multivalued context keys.

- **Policy variables** – Supported
- **Wildcards** – Not supported

Condition operator	Description
`Bool`	Boolean matching
`ForAllValues:Bool`	Use with the Array of Bool data type. All of the booleans in the context key values must match the boolean values in your policy. To prevent `ForAllValues` operators from evaluating missing context keys or context keys with empty values as Allowed, you can include the Null condition operator in your policy.
`ForAnyValue:Bool`	Use with the Array of Bool data type. At least one of the booleans in the context key values must match the boolean values in your policy.

Example boolean condition operator

The following identity-based policy uses the `Bool` condition operator with the `aws:SecureTransport` key to deny replicating objects and object tags to the destination bucket and its contents if the request is not over SSL.

> **Important**
>
> This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "BooleanExample",
      "Action": "s3:ReplicateObject",
      "Effect": "Deny",
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ],
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"Bool": { "aws:SecureTranspo rt": "false" }`	`aws:SecureTransport: - false`	Match
`"Bool": { "aws:SecureTranspo rt": "false" }`	`aws:SecureTransport: - true`	No match
`"Bool": { "aws:SecureTranspo rt": "false" }`	No `aws:SecureTranspor t` in the request context.	No match

Binary condition operators

The `BinaryEquals` condition operator lets you construct `Condition` elements that test key values that are in binary format. It compares the value of the specified key byte for byte against a base-64 encoded representation of the binary value in the policy. If the key that you specify in a policy condition is not present in the request context, the values do not match.

- **Policy variables** – Not supported
- **Wildcards** – Not supported

```
"Condition" : {
  "BinaryEquals": {
    "key" : "QmluYXJ5VmFsdWVJbkJhc2U2NA=="
  }
}
```

Policy condition	Request context	Result
`"BinaryEquals": { "key" : "QmluYXJ5VmFsdWVJbkJhc2U2NA==" }`	`key: - QmluYXJ5VmFsdWVJbkJhc2U2NA==`	Match
`"BinaryEquals": { "key" : "QmluYXJ5VmFsdWVJbkJhc2U2NA==" }`	`key: - ASIAIOSFODNN7EXAMPLE`	No match
`"BinaryEquals": { "key" : "QmluYXJ5VmFsdWVJbkJhc2U2NA==" }`	No `key` in the request context.	No match

IP address condition operators

IP address condition operators let you construct `Condition` elements that restrict access based on comparing a key to an IPv4 or IPv6 address or range of IP addresses. You use these with the `aws:SourceIp` key. The value must be in the standard CIDR format (for example, 203.0.113.0/24 or 2001:DB8:1234:5678::/64). If you specify an IP address without the associated routing prefix, IAM uses the default prefix value of `/32`.

Some AWS services support IPv6, using :: to represent a range of 0s. To learn whether a service supports IPv6, see the documentation for that service.

- **Policy variables** – Not supported
- **Wildcards** – Not supported

Condition operator	Description
`IpAddress`	The specified IP address or range
`NotIpAddress`	All IP addresses except the specified IP address or range

Example IP address condition operator

The following statement uses the `IpAddress` condition operator with the `aws:SourceIp` key to specify that the request must come from the IP range 203.0.113.0 to 203.0.113.255.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "iam:*AccessKey*",
        "Resource": "arn:aws:iam::111122223333:user/*",
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "203.0.113.0/24"
            }
        }
    }
}
```

The `aws:SourceIp` condition key resolves to the IP address that the request originates from. If the requests originates from an Amazon EC2 instance, `aws:SourceIp` evaluates to the instance's public IP address.

If the key that you specify in a policy condition is not present in the request context, the values do not match. The `aws:SourceIp` key is always present in the request context, except when the requester uses a VPC endpoint to make the request. In this case, the condition returns `false` and the request is implicitly denied by this statement.

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"IpAddress": { "aws:SourceIp": "203.0.113.0/24" }`	`aws:SourceIp: - 203.0.113.1`	Match

Policy condition	Request context	Result
`"IpAddress": { "aws:SourceIp": "203.0.113.0/24" }`	`aws:SourceIp: - 198.51.100.1`	No match

The following example shows how to mix IPv4 and IPv6 addresses to cover all of your organization's valid IP addresses. We recommend that you update your organization's policies with your IPv6 address ranges in addition to IPv4 ranges you already have to ensure the policies continue to work as you make the transition to IPv6.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "someservice:*",
    "Resource": "*",
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": [
          "203.0.113.0/24",
          "2001:DB8:1234:5678::/64"
        ]
      }
    }
  }
}
```

The `aws:SourceIp` condition key works only in a JSON policy if you are calling the tested API directly as a user. If you instead use a service to call the target service on your behalf, the target service sees the IP address of the calling service rather than the IP address of the originating user. This can happen, for example, if you use AWS CloudFormation to call Amazon EC2 to construct instances for you. There is currently no way to pass the originating IP address through a calling

service to the target service for evaluation in a JSON policy. For these types of service API calls, do not use the `aws:SourceIp` condition key.

Amazon Resource Name (ARN) condition operators

Amazon Resource Name (ARN) condition operators let you construct `Condition` elements that restrict access based on comparing a key to an ARN. The ARN is considered a string.

- **Policy variables** – Supported
- **Wildcards** – Supported

Condition operator	Description
`ArnEquals` , `ArnLike`	Case-sensitive matching of the ARN. Each of the six colon-delimited components of the ARN is checked separately and each can include multi-character match wildcards (*) or single-character match wildcards (?). The `ArnEquals` and `ArnLike` condition operators behave identically.
`ArnNotEquals` , `ArnNotLike`	Negated matching for ARN. The `ArnNotEquals` and `ArnNotLike` condition operators behave identically.

Example ARN condition operator

The following resource-based policy example shows a policy attached to an Amazon SQS queue to which you want to send SNS messages. It gives Amazon SNS permission to send messages to the queue (or queues) of your choice, but only if the service is sending the messages on behalf of a particular Amazon SNS topic (or topics). You specify the queue in the `Resource` field, and the Amazon SNS topic as the value for the `SourceArn` key.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {
```

```
            "Service": "sns.amazonaws.com"
        },
        "Action": "SQS:SendMessage",
        "Resource": "arn:aws:sqs:us-east-1:123456789012:QUEUE-ID",
        "Condition": {
            "ArnEquals": {
                "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:TOPIC-ID"
            }
        }
    }
}
```

The aws:SourceArn key is present in the request context only if a resource triggers a service to call another service on behalf of the resource owner. If an IAM user attempts to perform this operation directly, the condition returns `false` and the request is implicitly denied by this statement.

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"ArnEquals": { "aws:SourceArn": "arn:aws:sns:us-west-2:123456789012:TOPIC-ID" }`	`aws:SourceArn: - arn:aws:sns:us-west-2:123456789012:TOPIC-ID`	Match
`"ArnEquals": { "aws:SourceArn": "arn:aws:sns:us-west-2:123456789012:TOPIC-ID" }`	`aws:SourceArn: - arn:aws:sns:us-west-2:777788889999:TOPIC-ID`	No match
`"ArnEquals": { "aws:SourceArn": "arn:aws:sns:us-we`	No `aws:SourceArn` in the request context.	No match

Policy condition	Request context	Result
`st-2:123456789012:TOPIC-ID"}`		

Multivalued ARN condition operators

If a key in the request contains multiple values, ARN operators can be qualified with set operators `ForAllValues` and `ForAnyValue`. For more information on the evaluation logic of multiple context keys or values, see Set operators for multivalued context keys.

Condition operator	Description
`ForAllValues:ArnEquals` `ForAllValues:ArnLike`	All of the ARNs in the request context must match at least one of the ARN patterns in your policy.
`ForAnyValue:ArnEquals` `ForAnyValue:ArnLike`	At least one ARN in the request context must match one of the ARN patterns in your policy.
`ForAllValues:ArnNotEquals` `ForAllValues:ArnNotLike`	Negated matching. None of the ARNs in the request context can match any string ARN patterns in your policy.
`ForAnyValue:ArnNotEquals` `ForAnyValue:ArnNotLike`	Negated matching. At least one ARN in the request context must NOT match any of ARN patterns in your policy.

Example using `ForAllValues` with an ARN condition operator

The following example uses `ForAllValues:ArnLike` to create or update a logical delivery source for Amazon CloudWatch Logs logs. The condition block includes the condition key `logs:LogGeneratingResourceArns` to filter the log generating resource ARNs passed in the

request. Using this condition operator, all of the ARNs in the request must match at least one ARN in the policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "logs:PutDeliverySource",
            "Resource": "arn:aws:logs:us-east-1:123456789012:delivery-source:*",
            "Condition": {
                "ForAllValues:ArnLike": {
                    "logs:LogGeneratingResourceArns": [
                        "arn:aws:cloudfront::123456789012:distribution/*",
                        "arn:aws:cloudfront::123456789012:distribution/support*"
                    ]
                }
            }
        }
    ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"ForAllValues:ArnLike": { "logs:LogGeneratingResourceArns": [ "arn:aws::cloudfront:123456789012:distribution/*", "arn:aws::cloudfront:123456789012:distribution/support*"`	`logs:LogGeneratingResourceArns: - arn:aws::cloudfront:123456789012:distribution/costcenter`	Match

Policy condition	Request context	Result
] }		

"ForAllValues:ArnLike": {
"logs:LogGeneratingResourceArns": [
"arn:aws::cloudfront:123456789012:distribution/*",
"arn:aws::cloudfront:123456789012:distribution/support*"
]
}
	logs:LogGeneratingResourceArns:
	- arn:aws::cloudfront:123456789012:distribution/costcenter
	- arn:aws::cloudfront:123456789012:distribution/support2025
		Match

"ForAllValues:ArnLike": {
"logs:LogGeneratingResourceArns": [
"arn:aws::cloudfront:123456789012:distribution/*",
"arn:aws::cloudfront:123456789012:distribution/support*"
]
}
	logs:LogGeneratingResourceArns:
	- arn:aws::cloudfront:123456789012:distribution/costcenter
	- arn:aws::cloudfront:123456789012:distribution/admin
		Match

Policy condition	Request context	Result
`"ForAllValues:ArnLike": { "logs:LogGeneratingResourceArns": [ "arn:aws::cloudfront:123456789012:distribution/*", "arn:aws::cloudfront:123456789012:distribution/support*" ] }`	`logs:LogGeneratingResourceArns: - arn:aws::cloudfront:777788889999:distribution/costcenter`	No match
`"ForAllValues:ArnLike": { "logs:LogGeneratingResourceArns": [ "arn:aws::cloudfront:123456789012:distribution/*", "arn:aws::cloudfront:123456789012:distribution/support*" ] }`	No `logs:LogGeneratingResourceArns` in the request context.	Match

The `ForAllValues` qualifier returns true if there are no context keys in the request or if the context key value resolves to a null dataset, such as an empty string. To prevent missing context keys or context keys with empty values from evaluating to true, you can include the Null condition operator in your policy with a `false` value to check if the context key exists and its value is not null.

...IfExists condition operators

You can add `IfExists` to the end of any condition operator name except the `Null` condition —for example, `StringLikeIfExists`. You do this to say "If the condition key is present in the context of the request, process the key as specified in the policy. If the key is not

present, evaluate the condition element as true." Other condition elements in the statement can still result in a nonmatch, but not a missing key when checked with `...IfExists`. If you are using an `"Effect": "Deny"` element with a negated condition operator like `StringNotEqualsIfExists`, the request is still denied even if the condition key is not present.

Example using `IfExists`

Many condition keys describe information about a certain type of resource and only exist when accessing that type of resource. These condition keys are not present on other types of resources. This doesn't cause an issue when the policy statement applies to only one type of resource. However, there are cases where a single statement can apply to multiple types of resources, such as when the policy statement references actions from multiple services or when a given action within a service accesses several different resource types within the same service. In such cases, including a condition key that applies to only one of the resources in the policy statement can cause the `Condition` element in the policy statement to fail such that the statement's `"Effect"` does not apply.

For example, consider the following policy example:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "THISPOLICYDOESNOTWORK",
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": "*",
    "Condition": {"StringLike": {"ec2:InstanceType": [
      "t1.*",
      "t2.*",
      "m3.*"
    ]}}
  }
}
```

The *intent* of the preceding policy is to enable the user to launch any instance that is type `t1`, `t2` or `m3`. However, launching an instance requires accessing many resources in addition to the instance itself; for example, images, key pairs, security groups, and more. The entire statement is evaluated

against every resource that is required to launch the instance. These additional resources do not have the `ec2:InstanceType` condition key, so the `StringLike` check fails, and the user is not granted the ability to launch *any* instance type.

To address this, use the `StringLikeIfExists` condition operator instead. This way, the test only happens if the condition key exists. You could read the following policy as: "If the resource being checked has an "`ec2:InstanceType`" condition key, then allow the action only if the key value begins with `t1.`, `t2.`, or `m3.`. If the resource being checked does not have that condition key, then don't worry about it." The asterisk (*) in the condition key values, when used with the `StringLikeIfExists` condition operator, is interpreted as a wildcard to achieve partial string matches. The `DescribeActions` statement includes the actions required to view the instance in the console.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "RunInstance",
      "Effect": "Allow",
      "Action": "ec2:RunInstances",
      "Resource": "*",
      "Condition": {
        "StringLikeIfExists": {
          "ec2:InstanceType": [
            "t1.*",
            "t2.*",
            "m3.*"
          ]
        }
      }
    },
    {
      "Sid": "DescribeActions",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeImages",
        "ec2:DescribeInstances",
        "ec2:DescribeVpcs",
        "ec2:DescribeKeyPairs",
```

```
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"StringLikeIfExists": { "ec2:InstanceType": [ "t1.*", "t2.*", "m3.*" ] }`	`ec2:InstanceType: - t1.micro`	Match
`"StringLikeIfExists": { "ec2:InstanceType": [ "t1.*", "t2.*", "m3.*" ] }`	`ec2:InstanceType: - m2.micro`	No match
`"StringLikeIfExists": { "ec2:InstanceType": [ "t1.*", "t2.*",`	No `ec2:InstanceType` in the request context.	Match

Policy condition	Request context	Result
"m3.*"] }		

Condition operator to check existence of condition keys

Use a `Null` condition operator to check if a condition key is absent at the time of authorization. In the policy statement, use either `true` (the key doesn't exist — it is null) or `false` (the key exists and its value is not null).

You can not use a policy variable with the `Null` condition operator.

For example, you can use this condition operator to determine whether a user is using temporary credentials or their own credentials to make a request. If the user is using temporary credentials, then the key `aws:TokenIssueTime` exists and has a value. The following example shows a condition that states that the user must be using temporary credentials (the key cannot be absent) for the user to use the Amazon EC2 API.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":{
      "Action":"ec2:*",
      "Effect":"Allow",
      "Resource":"*",
      "Condition":{"Null":{"aws:TokenIssueTime":"false"}}
  }
}
```

Conditions with multiple context keys or values

You can use the `Condition` element of a policy to test multiple context keys or multiple values for a single context key in a request. When you make a request to AWS, either programmatically or through the AWS Management Console, your request includes information about your principal, operation, tags, and more. You can use context keys to test the values of the matching context keys

in the request, with the context keys specified in the policy condition. To learn about information and data included in a request, see The request context.

Topics

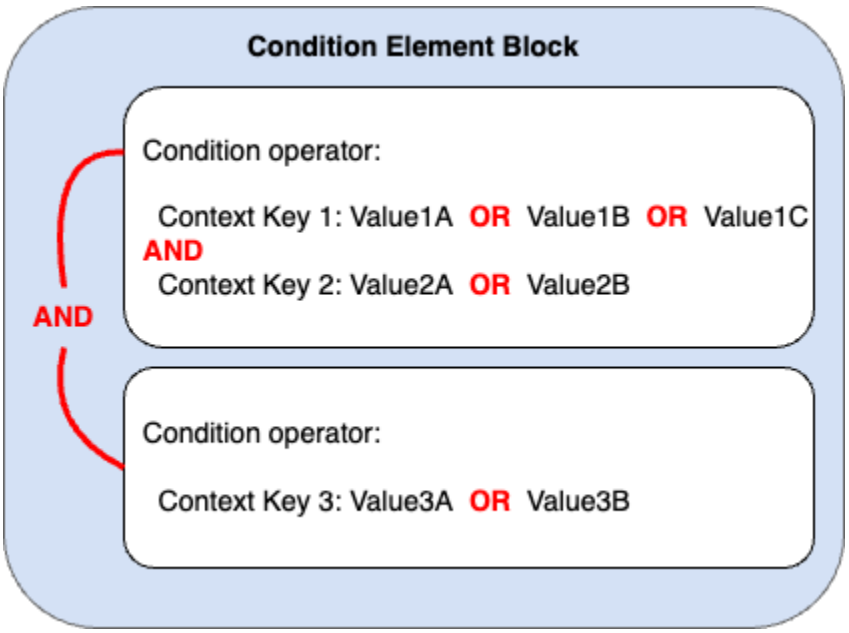
- Evaluation logic for multiple context keys or values
- Evaluation logic for negated matching condition operators

Evaluation logic for multiple context keys or values

A `Condition` element can contain multiple condition operators, and each condition operator can contain multiple context key-value pairs. Most context keys support using multiple values, unless otherwise specified.

- If your policy statement has multiple condition operators, the condition operators are evaluated using a logical `AND`.
- If your policy statement has multiple context keys attached to a single condition operator, the context keys are evaluated using a logical `AND`.
- If a single condition operator includes multiple values for a context key, those values are evaluated using a logical `OR`.
- If a single negated matching condition operator includes multiple values for a context key, those values are evaluated using a logical `NOR`.

All context keys in a condition element block must resolve to true to invoke the desired `Allow` or `Deny` effect. The following figure illustrates the evaluation logic for a condition with multiple condition operators and context key-value pairs.



For example, the following S3 bucket policy illustrates how the previous figure is represented in a policy. The condition block includes condition operators `StringEquals` and `ArnLike`, and context keys `aws:PrincipalTag` and `aws:PrincipalArn`. To invoke the desired `Allow` or `Deny` effect, all context keys in the condition block must resolve to true. The user making the request must have both principal tag keys, *department* and *role*, that include one of the tag key values specified in the policy. Also, the principal ARN of the user making the request must match one of the `aws:PrincipalArn` values specified in the policy to be evaluated as true.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ExamplePolicy",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::222222222222:root"
      },
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/department": [
            "finance",
```

```
            "hr",
            "legal"
          ],
          "aws:PrincipalTag/role": [
            "audit",
            "security"
          ]
        },
        "ArnLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::222222222222:user/Ana",
            "arn:aws:iam::222222222222:user/Mary"
          ]
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy Condition	Request Context	Result
`"StringEquals": { "aws:PrincipalTag/department": [ "finance", "hr", "legal" ], "aws:PrincipalTag/role": [ "audit", "security" ] }, "ArnLike": { "aws:PrincipalArn": [`	`aws:PrincipalTag/department: legal aws:PrincipalTag/role: audit aws:PrincipalArn: arn:aws:iam::222222222222:user/Mary`	**Match**

Policy Condition	Request Context	Result

`"arn:aws:iam::222222222222:user/Ana",`
`"arn:aws:iam::222222222222:user/Mary"`
`]`
`}`		

`"StringEquals": {`
`"aws:PrincipalTag/department": [`
`"finance",`
`"hr",`
`"legal"`
`],`
`"aws:PrincipalTag/role": [`
`"audit",`
`"security"`
`]`
`},`
`"ArnLike": {`
`"aws:PrincipalArn": [`
`"arn:aws:iam::222222222222:user/Ana",`
`"arn:aws:iam::222222222222:user/Mary"`
`]`
`}`
	`aws:PrincipalTag/department: hr`
	`aws:PrincipalTag/role: audit`
	`aws:PrincipalArn: arn:aws:iam::222222222222:user/ Nikki`
		No match

Policy Condition	Request Context	Result

"StringEquals": {
  "aws:PrincipalTag/
department": [
  "finance",
  "hr",
  "legal"
  ],
  "aws:PrincipalTag/
role": [
  "audit",
  "security"
  ]
},
"ArnLike": {
  "aws:PrincipalArn":
 [
  "arn:aws:
iam::222222222222:
user/Ana",
  "arn:aws:
iam::222222222222:
user/Mary"
  ]
}
	aws:PrincipalTag/d
	epartment: hr
	aws:PrincipalTag/
	role: *payroll*
	aws:PrincipalArn:
	  arn:aws:iam::22222
	2222222:user/Mary
		No match

Policy Condition	Request Context	Result

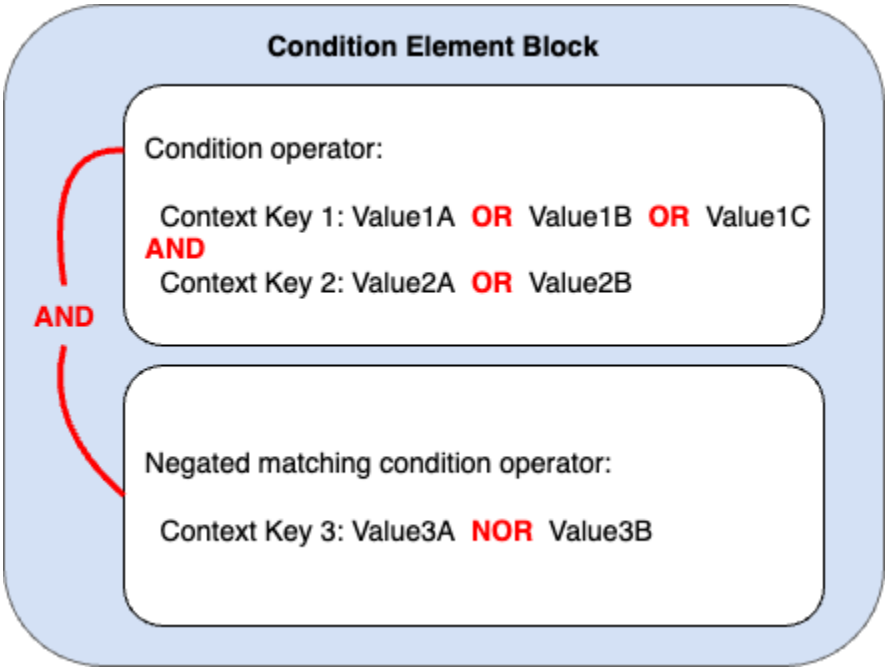
`"StringEquals": { "aws:PrincipalTag/department": [ "finance", "hr", "legal" ], "aws:PrincipalTag/role": [ "audit", "security" ] }, "ArnLike": { "aws:PrincipalArn": [ "arn:aws:iam::222222222222:user/Ana", "arn:aws:iam::222222222222:user/Mary" ] }`	No `aws:PrincipalTag/role` in the request context.
	`aws:PrincipalTag/department: hr`
	`aws:PrincipalArn: arn:aws:iam::222222222222:user/Mary`
		No match

Policy Condition	Request Context	Result
`"StringEquals": { "aws:PrincipalTag/department": [ "finance", "hr", "legal" ], "aws:PrincipalTag/role": [ "audit", "security" ] }, "ArnLike": { "aws:PrincipalArn": [ "arn:aws:iam::222222222222:user/Ana", "arn:aws:iam::222222222222:user/Mary" ] }`	No `aws:PrincipalTag` in the request context. `aws:PrincipalArn: arn:aws:iam::22222222222222:user/Mary`	**No match**

Evaluation logic for negated matching condition operators

Some condition operators, such as `StringNotEquals` or `ArnNotLike`, use negated matching to compare the context key-value pairs in your policy against the context key-value pairs in a request. When multiple values are specified for a single context key in a policy with negated matching condition operators, the effective permissions work like a logical `NOR`. In negated matching, a logical `NOR` or `NOT OR` returns true only if all values evaluate to false.

The following figure illustrates the evaluation logic for a condition with multiple condition operators and context key-value pairs. The figure includes a negated matching condition operator for context key 3.



For example, the following S3 bucket policy illustrates how the previous figure is represented in a policy. The condition block includes condition operators `StringEquals` and `ArnNotLike`, and context keys `aws:PrincipalTag` and `aws:PrincipalArn`. To invoke the desired `Allow` or `Deny` effect, all context keys in the condition block must resolve to true. The user making the request must have both principal tag keys, *department* and *role*, that include one of the tag key values specified in the policy. Since the `ArnNotLike` condition operator uses negated matching, the principal ARN of the user making the request must not match any of the `aws:PrincipalArn` values specified in the policy to be evaluated as true.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ExamplePolicy",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::222222222222:root"
      },
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Condition": {
```

```
        "StringEquals": {
          "aws:PrincipalTag/department": [
            "finance",
            "hr",
            "legal"
          ],
          "aws:PrincipalTag/role": [
            "audit",
            "security"
          ]
        },
        "ArnNotLike": {
          "aws:PrincipalArn": [
            "arn:aws:iam::222222222222:user/Ana",
            "arn:aws:iam::222222222222:user/Mary"
          ]
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy Condition	Request Context	Result
`"StringEquals": { "aws:PrincipalTag/department": [ "finance", "hr", "legal" ], "aws:PrincipalTag/role": [ "audit", "security" ] }, "ArnNotLike": {`	`aws:PrincipalTag/department: legal aws:PrincipalTag/role: audit aws:PrincipalArn: arn:aws:iam::222222222222:user/Nikki`	**Match**

Policy Condition	Request Context	Result
"aws:PrincipalArn": ["arn:aws:iam::222222222222:user/Ana", "arn:aws:iam::222222222222:user/Mary"] }		
"StringEquals": { "aws:PrincipalTag/department": ["finance", "hr", "legal"], "aws:PrincipalTag/role": ["audit", "security"] }, "ArnNotLike": { "aws:PrincipalArn": ["arn:aws:iam::222222222222:user/Ana", "arn:aws:iam::222222222222:user/Mary"] }	aws:PrincipalTag/department: hr aws:PrincipalTag/role: audit aws:PrincipalArn: arn:aws:iam::222222222222:user/ *Mary*	**No match**

Policy Condition	Request Context	Result

"StringEquals": {
  "aws:PrincipalTag/
department": [
  "finance",
  "hr",
  "legal"
  ],
  "aws:PrincipalTag/
role": [
  "audit",
  "security"
  ]
},
"ArnNotLike": {
  "aws:PrincipalArn":
[
  "arn:aws:
iam::222222222222:
user/Ana",
  "arn:aws:
iam::222222222222:
user/Mary"
  ]
}
	aws:PrincipalTag/d
	epartment: hr
	aws:PrincipalTag/
	role: *payroll*
	aws:PrincipalArn:
	  arn:aws:iam::22222
	2222222:user/Nikki
		No match

Policy Condition	Request Context	Result

`"StringEquals": { "aws:PrincipalTag/department": [ "finance", "hr", "legal" ], "aws:PrincipalTag/role": [ "audit", "security" ] }, "ArnNotLike": { "aws:PrincipalArn": [ "arn:aws:iam::222222222222:user/Ana", "arn:aws:iam::222222222222:user/Mary" ] }`	>
	 No `aws:PrincipalTag/role` in the request context.
	 `aws:PrincipalTag/department: hr`
	 `aws:PrincipalArn: arn:aws:iam::222222222222:user/Nikki`
		No match

Policy Condition	Request Context	Result
`"StringEquals": { "aws:PrincipalTag/department": [ "finance", "hr", "legal" ], "aws:PrincipalTag/role": [ "audit", "security" ] }, "ArnNotLike": { "aws:PrincipalArn": [ "arn:aws:iam::222222222222:user/Ana", "arn:aws:iam::222222222222:user/Mary" ] }`	No `aws:PrincipalTag` in the request context. `aws:PrincipalArn: arn:aws:iam::22222222222222:user/Nikki`	**No match**

Single-valued vs. multivalued context keys

The difference between single-valued and multivalued context keys lies in the number of values in the request context, not the number of values in the policy condition.

- *Single-valued* condition context keys have at most one value in the request context. For example, when you tag resources in AWS, each resource tag is stored as a key-value pair. Since a resource tag key can have only a single tag value, the section called “ResourceTag” is a single-valued context key. Do not use a condition set operator with a single-valued context key.
- *Multivalued* condition context keys can have multiple values in the request context. For example, when you tag resources in AWS, you can include multiple tag key-value pairs in a single request.

Therefore, the section called "TagKeys" is a multivalued context key. Multivalued context keys require a condition set operator.

For example, a request can originate from at most one VPC endpoint, so the section called "SourceVpce" is a single-valued context key. Since a service can have more than one service principal name that belongs to the service, the section called "PrincipalServiceNamesList" is a multivalued context key.

> **Important**
>
> The difference between single-valued and multivalued context keys depends on the number of values in the request context, not the number of values in the policy condition.

For policy examples that demonstrate conditions with multiple context keys and values, see Condition policy examples.

Key points

- The *Single-valued* and *Multivalued* classifications are included in the description of each condition context key as *Value type* in the AWS global condition context keys topic.
- Multivalued context keys in the Service Authorization Reference use an `ArrayOf` prefix followed by the condition operator category type, such as `ArrayOfString` or `ArrayOfARN`, indicating that the request may include multiple values for a condition context key.
- You can use any available single-valued context key as a policy variable, but you cannot use a multivalued context key as a policy variable. For more information about policy variables, see IAM policy elements: Variables and tags.
- When using context keys that include key-value pairs, it's important to note that even though there can be multiple tag-key values, each `tag-key` can have only one value.
  - the section called "PrincipalTag", the section called "RequestTag" and the section called "ResourceTag" are single-valued context keys.
  - the section called "TagKeys" defines what tag-keys are allowed in a request but does not include the tag-key values. Because you can include multiple tag key-value pairs in a request, `aws:TagKeys` is a multivalued context key.

- Multivalued context keys require a condition set operator. Do not use condition set operators `ForAllValues` or `ForAnyValue` with single-valued context keys. Using condition set operators with single-valued context keys can lead to overly permissive policies.

Set operators for multivalued context keys

To compare your condition context key against a request context key with multiple values, you must use the `ForAllValues` or `ForAnyValue` set operators. These set operators are used to compare two sets of values, such as the set of tags in a request and the set of tags in a policy condition.

The `ForAllValues` and `ForAnyValue` qualifiers add set-operation functionality to the condition operator, allowing you to test request context keys with multiple values against multiple context key values in a policy condition. Additionally, if you include a multivalued string context key in your policy with a wildcard or a variable, you must also use the `StringLike` condition operator. Multiple condition key values must be enclosed in brackets like an array, for example, `"Key2": ["Value2A", "Value2B"]`.

ForAllValues

The `ForAllValues` qualifier tests whether the value of every member of the request context matches the condition operator that follows the qualifier. The condition returns `true` if every context key value in the request matches a context key value in the policy. It also returns `true` if there are no context keys in the request.

> **Important**
>
> Use caution if you use `ForAllValues` with an `Allow` effect, as it can be overly permissive if the presence of missing context keys in the request context is unexpected. You should always include the `Null` condition operator in your policy with a `false` value to check if the context key exists and its value is not null. For an example, see Controlling access based on tag keys.

Example ForAllValues set operator

In the following example, ForAllValues is used with aws:TagKeys to allow users to delete specific tags assigned to an EC2 instance. This policy allows users to delete only the `environment` and

`cost-center` tags. You can delete them separately or together. The tag-keys in the request must match exactly the specified keys in the policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:DeleteTags",
            "Resource": "arn:aws:ec2:us-east-1:111122223333:instance/*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "environment",
                        "cost-center"
                    ]
                },
                "Null": {
                    "aws:TagKeys": "false"
                }
            }
        }
    ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy Condition	Request Context	Result
`"ForAllValues:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }, "Null": {`	`aws:TagKeys: - environment`	**Match**

Policy Condition	Request Context	Result
`"aws:TagKeys": "false" }`		
`"ForAllValues:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }, "Null": { "aws:TagKeys": "false" }`	`aws:TagKeys: - cost-center`	**Match**
`"ForAllValues:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }, "Null": { "aws:TagKeys": "false" }`	`aws:TagKeys: - environment - cost-center`	**Match**

Policy Condition	Request Context	Result
`"ForAllValues:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }, "Null": { "aws:TagKeys": "false" }`	`aws:TagKeys: - environment - dept`	**No match**
`"ForAllValues:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }, "Null": { "aws:TagKeys": "false" }`	No `aws:TagKeys` in the request context.	**No match**

Note that in the last example, the result is "No Match" because the Null condition check prevents matching when the context key is missing. This is a best practice to avoid overly permissive policies.

ForAnyValue

The `ForAnyValue` qualifier tests whether at least one member of the set of request context key values matches at least one member of the set of context key values in your policy condition. The condition returns `true` if any one of the context key values in the request matches any one of the context key values in the policy. For no matching context key or if the key does not exist, the condition returns `false`.

> **Important**
>
> When using `ForAnyValue` with a `Deny` effect, if the context key is not present in the request, the policy evaluates as **No match**. For consistent behavior, add an explicit `Null` condition check in your policy to verify whether the context key exists. For details, see Condition operator to check existence of condition keys.

Example ForAnyValue set operator

In the following example, ForAnyValue is used with aws:TagKeys to allow users to delete specific tags assigned to an EC2 instance. This policy allows users to delete tags for an instance if the tag keys specified in the request include `environment` or `cost-center`. The request can include additional tag keys beyond those specified in the policy, but must include at least one of the specified keys to match the condition.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:DeleteTags",
            "Resource": "arn:aws:ec2:us-east-1:111122223333:instance/*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": [
                        "environment",
                        "cost-center"
                    ]
                }
            }
        }
    ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy Condition	Request Context	Result
`"ForAnyValue:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }`	`aws:TagKeys: - environment`	**Match**
`"ForAnyValue:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }`	`aws:TagKeys: - cost-center`	**Match**
`"ForAnyValue:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }`	`aws:TagKeys: - environment - cost-center`	**Match**
`"ForAnyValue:StringEquals": { "aws:TagKeys": [ "environment", "cost-center" ] }`	`aws:TagKeys: - environment - dept`	**Match**
`"ForAnyValue:StringEquals": { "aws:TagKeys": [ "environment", "cost-center"`	`aws:TagKeys: - dept`	**No match**

Policy Condition	Request Context	Result
] }		

"ForAnyValue:StringEquals": {
"aws:TagKeys": [
"environment",
"cost-center"
]
}
	No aws:TagKeys in the request context.	**No match**

Condition policy examples

In IAM policies, you can specify multiple values for both single-valued and multivalued context keys for comparison against the request context. The following set of policy examples demonstrates policy conditions with multiple context keys and values.

> **Note**
>
> If you would like to submit a policy to be included in this reference guide, use the **Feedback** button at the bottom of this page. For IAM identity-based policy examples, see Example IAM identity-based policies.

Condition policy examples: Single-valued context keys

- Multiple condition blocks with single-valued context keys
- One condition block with multiple single-valued context keys and values

Condition policy examples: Multivalued context keys

- Deny policy with condition set operator ForAllValues
- Deny policy with condition set operator ForAnyValue

Multivalued context key examples

The following set of policy examples demonstrate how to create policy conditions with multivalued context keys.

Example: Deny policy with condition set operator ForAllValues

The following examples show how to use an identity-based policy to deny the use of IAM tagging actions when specific tag key prefixes are included in the request. The values for `aws:TagKeys` include a wildcard (*) for partial string matching. The policy includes the `ForAllValues` set operator with context key `aws:TagKeys` because the request context key can include multiple values. In order for context key `aws:TagKeys` to match, every value in the request context must match at least one value in the policy.

The `ForAllValues` set operator also returns true if there are no context keys in the request.

You can prevent missing context keys or context keys with empty values from evaluating to true by including a `Null` condition operator in your policy with a value of `false` to check if the context key in the request exists and its value is not null. For more information, see Condition operator to check existence of condition keys.

> **Important**
>
> This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

Example Deny a single policy condition value for a multivalued context key

In the following example, the policy denies requests where the values for `aws:TagKeys` in the request do not include the prefix **key1**. The request context can have multiple values, but because of the `ForAllValues` condition set operator, all the tag key values in the request context must start with the prefix **key1**.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "DenyRestrictedTags",
      "Effect": "Deny",
      "Action": [
        "iam:Tag*",
        "iam:UnTag*"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "ForAllValues:StringNotLike": {
          "aws:TagKeys": "key1*"
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request. For a Deny statement, Match is Denied and No match is Not denied, so it may be allowed by another statement.

Policy Condition	Request Context	Result
`"ForAllValues:StringNotLike": { "aws:TagKeys": "key1*" }`	`aws:TagKeys: - key1:legal`	**No match** May be allowed by another statement.
`"ForAllValues:StringNotLike": { "aws:TagKeys": "key1*" }`	`aws:TagKeys: - key1:hr - key1:personnel`	**No match** May be allowed by another statement.
`"ForAllValues:StringNotLike": {`	`aws:TagKeys: - key2:audit`	**Match**

Policy Condition	Request Context	Result
`"aws:TagKeys": "key1*" }`		
`"ForAllValues:StringNotLike": { "aws:TagKeys": "key1*" }`	No `aws:TagKeys` in the request context.	**Match**

Example Deny multiple policy condition values for a multivalued context key

In the following example, the policy denies requests where the values for `aws:TagKeys` in the request do not include the prefix **key1** or **key2**. The request context can have multiple values, but because of the `ForAllValues` condition set operator, all the tag key values in the request context must start with the prefix **key1** or **key2**.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyRestrictedTags",
      "Effect": "Deny",
      "Action": [
        "iam:Tag*",
        "iam:UnTag*"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "ForAllValues:StringNotLike": {
          "aws:TagKeys": [
            "key1*",
            "key2*"
          ]
```

```
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request. For a Deny statement, Match is Denied and No match is Not denied, so it may be allowed by another statement.

Policy Condition	Request Context	Result
`"ForAllValues:StringNotLike": { "aws:TagKeys": [ "key1*", "key2*" ] }`	`aws:TagKeys: - key1:legal`	**No match** May be allowed by another statement.
`"ForAllValues:StringNotLike": { "aws:TagKeys": [ "key1*", "key2*" ] }`	`aws:TagKeys: - key1:hr - key1:personnel`	**No match** May be allowed by another statement.
`"ForAllValues:StringNotLike": { "aws:TagKeys": [ "key1*", "key2*" ] }`	`aws:TagKeys: - key1:hr - key2:audit`	**No match** May be allowed by another statement.
`"ForAllValues:StringNotLike": {`	`aws:TagKeys: - key3:legal`	**Match**

Policy Condition	Request Context	Result
`"aws:TagKeys": [ "key1*", "key2*" ] }`		
`"ForAllValues:StringNotLike": { "aws:TagKeys": [ "key1*", "key2*" ] }`	No `aws:TagKeys` in the request context.	**Match**

Example: Deny policy with condition set operator ForAnyValue

The following identity-based policy example denies creating snapshots of EC2 instance volumes if any snapshots are tagged with one of the tag keys specified in the policy, `environment` or `webserver`. The policy includes the `ForAnyValue` set operator with context key `aws:TagKeys` because the request context key can include multiple values. If your tagging request includes any one of the tag key values specified in the policy, the `aws:TagKeys` context key returns true invoking the deny policy effect.

> **Important**
>
> This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
```

```
      "Action": [
        "ec2:CreateSnapshot",
        "ec2:CreateSnapshots"
      ],
      "Resource": "arn:aws:ec2:us-west-2::snapshot/*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:TagKeys": "webserver"
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request. For a Deny statement, Match is Denied and No match is Not denied, so it may be allowed by another statement.

Policy Condition	Request Context	Result
`"ForAnyValue:StringEquals": { "aws:TagKeys": "webserver" }`	`aws:TagKeys: - webserver`	**Match**
`"ForAnyValue:StringEquals": { "aws:TagKeys": "webserver" }`	`aws:TagKeys: - environment - webserver - test`	**Match**
`"ForAnyValue:StringEquals": { "aws:TagKeys": "webserver" }`	`aws:TagKeys: - environment - test`	**No match** May be allowed by another statement.

Policy Condition	Request Context	Result
`"ForAnyValue:StringEquals": { "aws:TagKeys": "webserver" }`	No `aws:TagKeys` in the request context.	**No match** May be allowed by another statement.

Single-valued context key policy examples

The following set of policy examples demonstrate how to create policy conditions with single-valued context keys.

Example: Multiple condition blocks with single-valued context keys

When a condition block has multiple conditions, each with a single context key, all context keys must resolve to true for the desired `Allow` or `Deny` effect to be invoked. When you use negated matching condition operators, the evaluation logic of the condition value is reversed.

The following example lets users create EC2 volumes and apply tags to the volumes during volume creation. The request context must include a value for context key `aws:RequestTag/project`, and the value for context key `aws:ResourceTag/environment` can be anything except production.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:CreateVolume",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ec2:CreateTags",
      "Resource": "arn:aws:ec2:us-east-1:123456789012:volume/*",
      "Condition": {
        "StringLike": {
```

```
          "aws:RequestTag/project": "*"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "ec2:CreateTags",
      "Resource": "arn:aws:ec2:us-east-1:123456789012:*/*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

The request context must include a project tag-value and cannot be created for a production resource to invoke the `Allow` effect. The following EC2 volume is successfully created because the project name is `Feature3` with a QA resource tag.

```
aws ec2 create-volume \
    --availability-zone us-east-1a \
    --volume-type gp2 \
    --size 80 \
    --tag-specifications 'ResourceType=volume,Tags=[{Key=project,Value=Feature3},
{Key=environment,Value=QA}]'
```

Example: One condition block with multiple single-valued context keys and values

When a condition block contains multiple context keys and each context key has multiple values, each context key must resolve to true for at least one key value for the desired `Allow` or `Deny` effect to be invoked. When you use negated matching condition operators, the evaluation logic of the context key value is reversed.

The following example allows users to start and run tasks on Amazon Elastic Container Service clusters.

- The request context must include `production` **OR** `prod-backup` for the `aws:RequestTag/environment` context key **AND**.

- The `ecs:cluster` context key makes sure that tasks are run on either the `default1` **OR** `default2` ARN ECS clusters.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:RunTask",
        "ecs:StartTask"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/environment": [
            "production",
            "prod-backup"
          ]
        },
        "ArnEquals": {
          "ecs:cluster": [
            "arn:aws:ecs:us-east-1:111122223333:cluster/default1",
            "arn:aws:ecs:us-east-1:111122223333:cluster/default2"
          ]
        }
      }
    }
  ]
}
```

The following table shows how AWS evaluates this policy based on the condition key values in your request.

Policy condition	Request context	Result
`"StringEquals": { "aws:RequestTag/environment": [ "production", "prod-backup" ] }, "ArnEquals": { "ecs:cluster": [ "arn:aws:ecs:us-east-1:111122223333:cluster/default1", "arn:aws:ecs:us-east-1:111122223333:cluster/default2" ] }`	`aws:RequestTag: environment:production ecs:cluster: arn:aws:ecs:us-east-1:111122223333:cluster/default1`	Match
`"StringEquals": { "aws:RequestTag/environment": [ "production", "prod-backup" ] }, "ArnEquals": { "ecs:cluster": [ "arn:aws:ecs:us-east-1:111122223333:cluster/default1", "arn:aws:ecs:us-east-1:111122223333:cluster/default2" ] }`	`aws:RequestTag: environment:prod-backup ecs:cluster: arn:aws:ecs:us-east-1:111122223333:cluster/default2`	Match
`"StringEquals": { "aws:RequestTag/environment": [`	`aws:RequestTag: webserver:production ecs:cluster:`	No match

Policy condition	Request context	Result

`"production",`
`"prod-backup"`
`]`
`},`
`"ArnEquals": {`
`"ecs:cluster": [`
`"arn:aws:ecs:us-east-1:111122223333:cluster/default1",`
`"arn:aws:ecs:us-east-1:111122223333:cluster/default2"`
`]`
`}`
	`arn:aws:ecs:us-east-1:111122223333:cluster/default2`	

`"StringEquals": {`
`"aws:RequestTag/environment": [`
`"production",`
`"prod-backup"`
`]`
`},`
`"ArnEquals": {`
`"ecs:cluster": [`
`"arn:aws:ecs:us-east-1:111122223333:cluster/default1",`
`"arn:aws:ecs:us-east-1:111122223333:cluster/default2"`
`]`
`}`
	No `aws:RequestTag` in the request context.
	`ecs:cluster`
	`arn:aws:ecs:us-east-1:111122223333:cluster/default2`
		No match

IAM policy elements: Variables and tags

Use AWS Identity and Access Management (IAM) policy variables as placeholders when you don't know the exact value of a resource or condition key when you write the policy.

> **Note**
>
> If AWS cannot resolve a variable this might cause the entire statement to be invalid. For example, if you use the `aws:TokenIssueTime` variable, the variable resolves to a value only when the requester authenticated using temporary credentials (an IAM role). To prevent variables from causing invalid statements, use the ...IfExists condition operator.

Topics

- Introduction
- Using variables in policies
- Tags as policy variables
- Where you can use policy variables
- Policy variables with no value
- Request information that you can use for policy variables
- Specifying default values
- For more information

Introduction

In IAM policies, many actions allow you to provide a name for the specific resources that you want to control access to. For example, the following policy allows users to list, read, and write objects in the S3 bucket `amzn-s3-demo-bucket` for `marketing` projects.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket"],
      "Condition": {"StringLike": {"s3:prefix": ["marketing/*"]}}
    },
    {
```

```
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/marketing/*"]
    }
  ]
}
```

In some cases, you might not know the exact name of the resource when you write the policy. You might want to generalize the policy so it works for many users without having to make a unique copy of the policy for each user. Instead of creating a separate policy for each user, we recommend you create a single group policy that works for any user in that group.

Using variables in policies

You can define dynamic values inside policies by using *policy variables* that set placeholders in a policy.

Variables are marked using a **$** prefix followed by a pair of curly braces (**{ }**) that include the variable name of the value from the request.

When the policy is evaluated, the policy variables are replaced with values that come from the conditional context keys passed in the request. Variables can be used in identity-based policies, resource policies, service control policies, session policies, and VPC endpoint policies. Identity-based policies used as permissions boundaries also support policy variables.

Global condition context keys can be used as variables in requests across AWS services. Service specific condition keys can also be used as variables when interacting with AWS resources, but are only available when requests are made against resources which support them. For a list of context keys available for each AWS service and resource, see the *Service Authorization Reference*. Under certain circumstances, you can't populate global condition context keys with a value. To learn more about each key, see AWS global condition context keys.

Important

- Key names are case-insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

- You can use any single-valued condition key as a variable. You can't use a multivalued condition key as a variable.

The following example shows a policy for an IAM role or user that replaces a specific resource name with a policy variable. You can reuse this policy by taking advantage of the `aws:PrincipalTag` condition key. When this policy is evaluated, `${aws:PrincipalTag/team}` allows the actions only if the bucket name ends with a team name from the `team` principal tag.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${aws:PrincipalTag/team}/*"]}}
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/${aws:PrincipalTag/team}/*"]
    }
  ]
}
```

The variable is marked using a $ prefix followed by a pair of curly braces ({ }). Inside the ${ } characters, you can include the name of the value from the request that you want to use in the policy. The values you can use are discussed later on this page.

For details about this global condition key, see aws:PrincipalTag/*tag-key* in the list of global condition keys.

Note

In order to use policy variables, you must include the `Version` element in a statement, and the version must be set to a version that supports policy variables. Variables were introduced in version `2012-10-17`. Earlier versions of the policy language don't support policy variables. If you don't include the `Version` element and set it to an appropriate version date, variables like `${aws:username}` are treated as literal strings in the policy. A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you change a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. To learn more about the `Version` policy element see the section called “Version”. To learn more about policy versions, see the section called “Versioning IAM policies”.

A policy that allows a principal to get objects from the /David path of an S3 bucket looks like this:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/David/*"
      ]
    }
  ]
}
```

If this policy is attached to user `David`, that user get objects from his own S3 bucket, but you would have to create a separate policy for each user that includes the user's name. You would then attach each policy to the individual users.

By using a policy variable, you can create policies that can be reused. The following policy allows a user to get objects from an Amazon S3 bucket if the tag-key value for `aws:PrincipalTag` matches the tag-key `owner` value passed in the request.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
    "Sid": "AllowUnlessOwnedBySomeoneElse",
    "Effect": "Allow",
    "Action": ["s3:GetObject"],
    "Resource": ["*"],
    "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/owner": "${aws:PrincipalTag/owner}"
        }
      }
    }
  ]
}
```

When you use a policy variable in place of a user like this, you don't have to have a separate policy for each individual user. In the following example, the policy is attached to an IAM role that is assumed by Product Managers using temporary security credentials. When a user makes a request to add an Amazon S3 object, IAM substitutes the `dept` tag value from the current request for the `${aws:PrincipalTag}` variable and evaluates the policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOnlyDeptS3Prefix",
```

```
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/${aws:PrincipalTag/dept}/*"
            ]
        }
    ]
}
```

Tags as policy variables

In some AWS services you can attach your own custom attributes to resources that are created by those services. For example, you can apply tags to Amazon S3 buckets or to IAM users. These tags are key-value pairs. You define the tag key name and the value that is associated with that key name. For example, you might create a tag with a **`department`** key and a **`Human Resources`** value. For more information about tagging IAM entities, see Tags for AWS Identity and Access Management resources. For information about tagging resources created by other AWS services, see the documentation for that service. For information about using Tag Editor, see Working with Tag Editor in the *AWS Management Console User Guide*.

You can tag IAM resources to simplify discovering, organizing, and tracking your IAM resources. You can also tag IAM identities to control access to resources or to tagging itself. To learn more about using tags to control access, see Controlling access to and for IAM users and roles using tags.

Where you can use policy variables

You can use policy variables in the `Resource` element and in string comparisons in the `Condition` element.

Resource element

You can use a policy variable in the `Resource` element, but only in the resource portion of the ARN. This portion of the ARN appears after the fifth colon (:). You can't use a variable to replace parts of the ARN before the fifth colon, such as the service or account. For more information about the ARN format, see IAM ARNs.

To replace part of an ARN with a tag value, surround the prefix and key name with `${ }`. For example, the following Resource element refers to only a bucket that is named the same as the value in the requesting user's department tag.

```
"Resource": ["arn:aws::s3:::amzn-s3-demo-bucket/
${aws:PrincipalTag/department}"]
```

Many AWS resources use ARNs that contain a user-created name. The following IAM policy ensures that only intended users with matching access-project, access-application, and access-environment tag values can modify their resources. In addition, using * wildcard matches, they are able to allow for custom resource name suffixes.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessBasedOnArnMatching",
      "Effect": "Allow",
      "Action": [
        "sns:CreateTopic",
        "sns:DeleteTopic"],
      "Resource": ["arn:aws:sns:*:*:${aws:PrincipalTag/access-project}-
${aws:PrincipalTag/access-application}-${aws:PrincipalTag/access-environment}-*"
      ]
    }
  ]
}
```

Condition element

You can use a policy variable for `Condition` values in any condition that involves the string operators or the ARN operators. String operators include `StringEquals`, `StringLike`, and `StringNotLike`. ARN operators include `ArnEquals` and `ArnLike`. You can't use a policy variable with other operators, such as `Numeric`, `Date`, `Boolean`, `Binary`, `IP Address`, or `Null` operators. For more information about condition operators, see IAM JSON policy elements: Condition operators.

When referencing a tag in a `Condition` element expression, use the relevant prefix and key name as the condition key. Then use the value that you want to test in the condition value.

For example, the following policy example allows full access to users, but only if the tag `costCenter` is attached to the user. The tag must also have a value of either `12345` or `67890`. If the tag has no value, or has any other value, then the request fails.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iam:*user*"
       ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iam:ResourceTag/costCenter": [ "12345", "67890" ]
        }
      }
    }
  ]
}
```

Policy variables with no value

When policy variables reference a condition context key that has no value or is not present in an authorization context for a request, the value is effectively null. There is no equal or like value. Condition context keys may not be present in the authorization context when:

- You are using service specific condition context keys in requests to resources that do not support that condition key.
- Tags on IAM principals, sessions, resources, or requests are not present.
- Other circumstances as listed for each global condition context key in AWS global condition context keys.

When you use a variable with no value in the condition element of an IAM policy, IAM JSON policy elements: Condition operators like `StringEquals` or `StringLike` do not match, and the policy statement does not take effect.

Inverted condition operators like `StringNotEquals` or `StringNotLike` do match against a null value, as the value of the condition key they are testing against is not equal to or like the effectively null value.

In the following example, `aws:principaltag/Team` must be equal to `s3:ExistingObjectTag/Team` to allow access. Access is explicitly denied when `aws:principaltag/Team` is not set. If a variable that has no value in the authorization context is used as part of the `Resource` or `NotResource` element of a policy, the resource that includes a policy variable with no value will not match any resource.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
   {
    "Effect": "Deny",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
      "StringNotEquals": {
        "s3:ExistingObjectTag/Team": "${aws:PrincipalTag/Team}"
       }
      }
    }
  ]
}
```

Request information that you can use for policy variables

You can use the `Condition` element of a JSON policy to compare keys in the request context with key values that you specify in your policy. When you use a policy variable, AWS substitutes a value from the request context key in place of the variable in your policy.

Principal key values

The values for `aws:username`, `aws:userid`, and `aws:PrincipalType` depend on what type of principal initiated the request. For example, the request could be made using the credentials of an IAM user, an IAM role, or the AWS account root user. The following list shows values for these keys for different types of principals.

- **AWS account root user**
  - `aws:username`: (not present)
  - `aws:userid`: AWS account ID
  - `aws:PrincipalType`: `Account`
- **IAM user**
  - `aws:username`: *`IAM-user-name`*
  - `aws:userid`: unique ID
  - `aws:PrincipalType`: `User`
- **AWS STS federated user principal**
  - `aws:username`: (not present)
  - `aws:userid`: *`account`*:*`caller-specified-name`*
  - `aws:PrincipalType`: `FederatedUser`
- **OIDC and SAML federated principals**

  > **Note**
  >
  > For information about policy keys that are available when you use OIDC federation, see OIDC federation.

  - `aws:username`: (not present)
  - `aws:userid`: (not present)
  - `aws:PrincipalType`: `AssumedRole`
- **Assumed role**
  - `aws:username`: (not present)
  - `aws:userid`: *`role-id`*:*`caller-specified-role-name`*
  - `aws:PrincipalType`: `Assumed role`

- **Role assigned to Amazon EC2 instance**
  - `aws:username`: (not present)
  - `aws:userid`: *`role-id`*:*`ec2-instance-id`*
  - `aws:PrincipalType`: `Assumed role`
- **Anonymous caller** (Amazon SQS Amazon SNS and Amazon S3 only)
  - `aws:username`: (not present)
  - `aws:userid`: (not present)
  - `aws:PrincipalType`: `Anonymous`

For the items in this list, note the following:

- *not present* means that the value is not in the current request information, and any attempt to match it fails and causes the statement to be invalid.
- *`role-id`* is a unique identifier assigned to each role at creation. You can display the role ID with the AWS CLI command: `aws iam get-role --role-name` *`rolename`*
- *`caller-specified-name`* and *`caller-specified-role-name`* are names that are passed by the calling process (such as an application or service) when it makes a call to get temporary credentials.
- *`ec2-instance-id`* is a value assigned to the instance when it is launched and appears on the **Instances** page of the Amazon EC2 console. You can also display the instance ID by running the AWS CLI command: `aws ec2 describe-instances`

Information available in requests for federated principals

Federated principals are users who are authenticated using a system other than IAM. For example, a company might have an application for use in-house that makes calls to AWS. It might be impractical to give an IAM identity to every corporate user who uses the application. Instead, the company might use a proxy (middle-tier) application that has a single IAM identity, or the company might use a SAML identity provider (IdP). The proxy application or SAML IdP authenticates individual users using the corporate network. A proxy application can then use its IAM identity to get temporary security credentials for individual users. A SAML IdP can in effect exchange identity information for AWS temporary security credentials. The temporary credentials can then be used to access AWS resources.

Similarly, you might create an app for a mobile device in which the app needs to access AWS resources. In that case, you might use *OIDC federation*, where the app authenticates the user using a well-known identity provider like Login with Amazon, Amazon Cognito, Facebook, or Google. The app can then use the user's authentication information from these providers to get temporary security credentials for accessing AWS resources.

The recommended way to use OIDC federation is by taking advantage of Amazon Cognito and the AWS mobile SDKs. For more information, see the following:

- Amazon Cognito User Guide
- Common scenarios for temporary credentials

Special characters

There are a few special predefined policy variables that have fixed values that enable you to represent characters that otherwise have special meaning. If these special characters are part of the string, you are trying to match and you inserted them literally they would be misinterpreted. For example, inserting an * asterisk in the string would be interpreted as a wildcard, matching any characters, instead of as a literal *. In these cases, you can use the following predefined policy variables:

- **${*}** - use where you need an * (asterisk) character.
- **${?}** - use where you need a ? (question mark) character.
- **${$}** - use where you need a $ (dollar sign) character.

These predefined policy variables can be used in any string where you can use regular policy variables.

Specifying default values

To add a default value to a variable, surround the default value with single quotes (`' '`), and separate the variable text and the default value with a comma and space (`, `).

For example, if a principal is tagged with `team=yellow`, they can access `ExampleCorp's` Amazon S3 bucket named `amzn-s3-demo-bucket-yellow`. A policy with this resource allows team members to access their team bucket, but not those of other teams. For users without team tags, it sets a default value of `company-wide` for the bucket name. These users can access only the

amzn-s3-demo-bucket-company-wide bucket where they can view broad information, such as instructions for joining a team.

```
"Resource":"arn:aws:s3:::amzn-s3-demo-bucket-${aws:PrincipalTag/team, 'company-wide'}"
```

For more information

For more information about policies, see the following:

- Policies and permissions in AWS Identity and Access Management
- Example IAM identity-based policies
- IAM JSON policy element reference
- Policy evaluation logic
- OIDC federation

IAM JSON policy elements: Supported data types

This section lists the data types that are supported when you specify values in JSON policies. The policy language doesn't support all types for each policy element; for information about each element, see the preceding sections.

- Strings
- Numbers (Ints and Floats)
- Boolean
- Null
- Lists
- Maps
- Structs (which are just nested Maps)

The following table maps each data type to the serialization. Note that all policies must be in UTF-8. For information about the JSON data types, see RFC 4627.

Type	JSON
String	String

Type	JSON
Integer	Number
Float	Number
Boolean	true false
Null	null
Date	String adhering to the W3C Profile of ISO 8601
IpAddress	String adhering to RFC 4632
List	Array
Object	Object

Policy evaluation logic

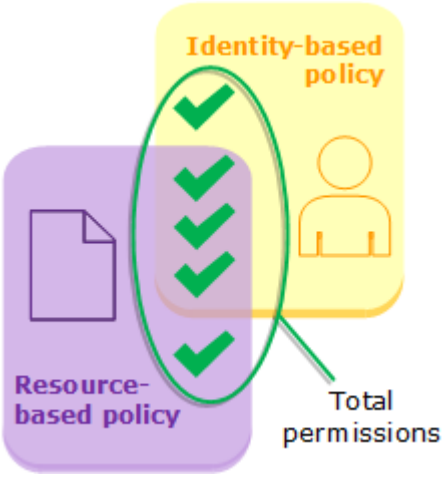
When a principal tries to use the AWS Management Console, the AWS API, or the AWS CLI, that principal sends a *request* to AWS. When an AWS service receives the request, AWS completes several steps to determine whether to allow or deny the request.

1. **Authentication** – AWS first authenticates the principal that makes the request, if necessary. This step is not necessary for a few services, such as Amazon S3, that allow some requests from anonymous users.
2. **Processing the request context** – AWS processes the information gathered in the request to determine which policies apply to the request.
3. **How AWS enforcement code logic evaluates requests to allow or deny access** – AWS evaluates all of the policy types and the order of the policies affects how they are evaluated. AWS then processes the policies against the request context to determine whether the request is allowed or denied.

Evaluating identity-based policies with resource-based policies

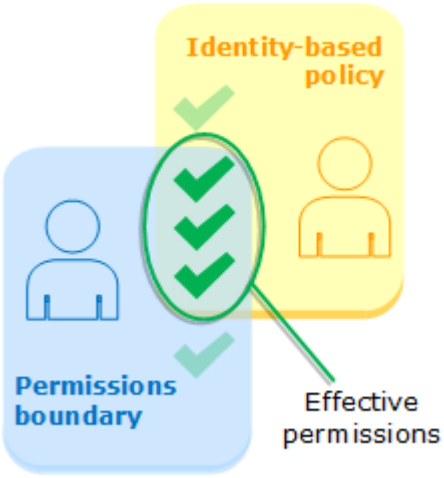
Identity-based policies and resource-based policies grant permissions to the identities or resources to which they are attached. When an IAM entity (user or role) requests access to a resource within

the same account, AWS evaluates all the permissions granted by the identity-based and resource-based policies. The resulting permissions are the union of the permissions of the two types. If an action is allowed by an identity-based policy, a resource-based policy, or both, then AWS allows the action. An explicit deny in either of these policies overrides the allow.



Evaluating identity-based policies with permissions boundaries

When AWS evaluates the identity-based policies and permissions boundary for a user, the resulting permissions are the intersection of the two categories. That means that when you add a permissions boundary to a user with existing identity-based policies, you might reduce the actions that the user can perform. Alternatively, when you remove a permissions boundary from a user, you might increase the actions they can perform. An explicit deny in either of these policies overrides the allow. To view information about how other policy types are evaluated with permissions boundaries, see Evaluating effective permissions with boundaries.



Evaluating identity-based policies with AWS Organizations SCPs or RCPs

When a user belongs to an account that is a member of an organization and accesses a resource that doesn't have a resource-based policy configured, the resulting permissions are the intersection of the user's policies, service control policies (SCPs), and resource control policy (RCP). This means that an action must be allowed by all three policy types. An explicit deny in the identity-based policy, an SCP, or an RCP overrides the allow.



You can learn whether your account is a member of an organization in AWS Organizations. Organization members might be affected by an SCP or RCP. To view this data using the AWS CLI command or AWS API operation, you must have permissions for the `organizations:DescribeOrganization` action for your AWS Organizations entity. You must have additional permissions to perform the operation in the AWS Organizations console. To learn whether an SCP or RCP is denying access to a specific request, or to change your effective permissions, contact your AWS Organizations administrator.

Processing the request context

When AWS evaluates and authorizes a request, it assembles the request information into a *request context*. The request context contains any information that can be used in policy evaluation.

- **Principal** – The user, role, or federated user principal that sent the request. Information about the principal includes the policies that are associated with that principal.
- **Actions** – One or more actions that the principal wants to perform.
- **Resources** – One or more AWS resource objects upon which the actions or operations are performed.

- **Resource data** – Data related to the resource that is being requested. This can include information such as a DynamoDB table name or a tag on an Amazon EC2 instance.
- **Environment data** – Information about the IP address, user agent, SSL enabled status, or the time of day.

This information is compared against applicable policies to determine whether to allow or deny the request. You can organize this property information using the **Principal**, **Action**, **Resource**, and **Condition** (PARC) model to better understand how AWS policies are evaluated.

Understanding the PARC model

The PARC model represents the request context based on the four JSON elements in the policy language:

- Principal – The entity making the request. A principal represents a human user or programmatic workload that can be authenticated and then authorized to perform actions in AWS accounts.
- Action – The operation being performed. Often the action will map to an API action.
- Resource – The AWS resource on which the action is being performed.
- Condition – Additional constraints that must be met for the request to be allowed.

The following shows an example of how the PARC model might represent a request context:

```
Principal: AIDA123456789EXAMPLE
Action: s3:CreateBucket
Resource: arn:aws:s3:::amzn-s3-demo-bucket1
Context:
- aws:UserId=AIDA123456789EXAMPLE:BobsSession
- aws:PrincipalAccount=123456789012
- aws:PrincipalOrgId=o-example
- aws:PrincipalARN=arn:aws:iam::AIDA123456789EXAMPLE:role/HR
- aws:MultiFactorAuthPresent=true
- aws:CurrentTime=...
- aws:EpochTime=...
- aws:SourceIp=...
- aws:PrincipalTag/dept=123
- aws:PrincipalTag/project=blue
- aws:RequestTag/dept=123
```

Importance of the request context

Understanding the request context and how it interacts with policy evaluation is crucial for:

- Troubleshooting access issues
- Designing effective and secure policies
- Understanding the full scope of permissions granted by a policy
- Predicting the outcome of policy evaluations in different scenarios

By visualizing the request context using the PARC model, you can more easily understand how AWS makes authorization decisions and design your policies more effectively.

How AWS uses the request context

When evaluating policies, AWS compares the information in the request context with the information specified in all applicable policies. This includes identity-based policies, resource-based policies, IAM permissions boundaries, Organizations SCPs, Organizations RCPs, and session policies.

For each policy type, AWS uses the request context to check:

- Whether the policy applies to the request based on the principal.
- Whether the requested action is allowed on the specified resource.
- Whether any conditions specified in the policy are met by the request context.

How AWS evaluates policies depends on the types of policies that apply to the request context. These policy types are available for use within a single AWS account. For more information about these policy types, see Policies and permissions in AWS Identity and Access Management. To learn how AWS evaluates policies for cross-account access, see Cross-account policy evaluation logic.

- **AWS Organizations resource control policies (RCPs)** – AWS Organizations RCPs specify the maximum available permissions for resources within accounts in an organization or organizational unit (OU). RCPs apply to resources in member accounts and impact the effective permissions for principals, including the AWS account root user, regardless of whether the principals belong to your organization. RCPs don't apply to resources in the organization management account and to calls made by service-linked roles. If an RCP is present, permissions granted by identity-based and resource-based policies to resources in your member accounts are only effective if the RCP allows the action.

- **AWS Organizations service control policies (SCPs)** – AWS Organizations SCPs specify the maximum available permissions for principals within accounts in an organization or organizational unit (OU). SCPs apply to principals in member accounts, including each AWS account root user. If an SCP is present, permissions granted by identity-based and resource-based policies to principals in your member accounts are only effective if the SCP allows the action. The only exceptions are principals in the organization management account and service-linked roles.
- **Resource-based policies** – Resource-based policies grant permissions for principals specified in the policy. The permissions define what the principal can do with the resource to which the policy is attached.
- **Permissions boundaries** – Permissions boundaries are a feature that sets the maximum permissions that an identity-based policy can grant to an IAM entity (user or role). When you set a permissions boundary for an entity, the entity can perform only the actions that are allowed by both its identity-based policies and its permissions boundary. In some cases, an implicit deny in a permissions boundary can limit the permissions granted by a resource-based policy. For more information, see How AWS enforcement code logic evaluates requests to allow or deny access.
- **Identity-based policies** – Identity-based policies are attached to an IAM identity (user, group of users, or role) and grant permissions to IAM entities (users and roles). If only identity-based policies apply to a request, then AWS checks all of those policies for at least one `Allow`.
- **Session policies** – Session policies are policies that you pass as parameters when you programmatically create a temporary session for a role or federated user session. To create a role session programmatically, use one of the `AssumeRole*` API operations. When you do this and pass session policies, the resulting session's permissions are the intersection of the IAM entity's identity-based policy and the session policies. To create a federated user session, you use the IAM user access keys to programmatically call the `GetFederationToken` API operation. For more information, see Session policies.

Remember, an explicit deny in any of these policies overrides the allow.

> **Note**
>
> **AWS Organizations declarative policies** allow you to centrally declare and enforce your desired configuration for a given AWS service at scale across an organization. Since declarative policies are applied directly at the service level, they don't directly impact policy

evaluation requests and aren't included with the request context. For more information, see Declarative policies in the AWS Organizations User Guide.

Example policy evaluation using the PARC model

To illustrate how the request context interacts with policy evaluation, let's consider an example policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:CreateBucket",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/dept": "123"
        }
      }
    }
  ]
}
```

In this example, the policy would allow the `CreateBucket` action only when the request context includes an `aws:PrincipalTag/dept` value of "123" and the resource matches the `amzn-s3-demo-bucket1` bucket name. The following table shows how AWS uses the request context to evaluate this policy and make authorization decisions.

Policy	Request context	Evaluation result
`"StringEquals": { "aws:PrincipalTag/dept": "123" }`	`Principal: AIDA123456789EXAMPLE` `Action: s3:CreateBucket`	**Match**

Policy	Request context	Evaluation result
	Resource: arn:aws:s3:::amzn-s3-demo-bucket1 Context: - aws:PrincipalTag/dept=123	

"StringEquals": {
"aws:PrincipalTag/dept": "123"
}
	Principal: AIDA123456789EXAMPLE
	Action: s3: *DeleteBucket*
	Resource: arn:aws:s3:::amzn-s3-demo-bucket1
	Context:
	- aws:PrincipalTag/dept=123
		No match

"StringEquals": {
"aws:PrincipalTag/dept": "123"
}
	Principal: AIDA123456789EXAMPLE
	Action: s3:CreateBucket
	Resource: arn:aws:s3:::amzn-s3-demo-bucket1
	Context:
	- aws:PrincipalTag/dept= *321*
		No match

Policy	Request context	Evaluation result

`"StringEquals": { "aws:PrincipalTag/dept": "123" }`	`Principal: AIDA12345 6789EXAMPLE`
	`Action: s3:Create Bucket`
	`Resource: arn:aws:s 3:::amzn-s3-demo-b ucket1`
	`Context:`
	
	No `aws:PrincipalTag` in the request.
		No match

How AWS enforcement code logic evaluates requests to allow or deny access

The AWS enforcement code decides whether a request sent to AWS should be allowed or denied. AWS evaluates all policies that are applicable to the request context. The following is a summary of the AWS policy evaluation logic.

- By default, all requests are implicitly denied with the exception of the AWS account root user, which has full access.
- Requests must be explicitly allowed by a policy or set of policies following the evaluation logic below to be allowed.
- An explicit deny overrides an explicit allow.

Policy evaluation can differ depending on whether the request is within a single account or a cross-account request. For details about how a policy evaluation decision is made for an IAM role or user within a single account, see Policy evaluation for requests within a single account. For details about how a policy evaluation decision is made for cross-account requests, see Cross-account policy evaluation logic.

- **Deny evaluation** – By default, all requests are denied. This is called an implicit deny. The AWS enforcement code evaluates all policies within the account that apply to the request. These include AWS Organizations SCPs and RCPs, resource-based policies, identity-based policies, IAM permissions boundaries, and session policies. In all those policies, the enforcement code looks for a Deny statement that applies to the request. This is called an explicit deny. If the enforcement

code finds even one explicit deny that applies, the enforcement code returns a final decision of **Deny**. If there is no explicit deny, the enforcement code evaluation continues.

- **AWS Organizations RCPs** – The enforcement code evaluates AWS Organizations resource control policies (RCPs) that apply to the request. RCPs apply to resources of the account where the RCPs are attached. If the enforcement code does not find any applicable `Allow` statements in the RCPs, the enforcement code returns a final decision of **Deny**. Note that an AWS managed policy called `RCPFullAWSAccess` is automatically created and attached to every entity in your organization including the root, each OU, and AWS account when RCPs are enabled. `RCPFullAWSAccess` can't be detached, so there will always be an `Allow` statement. If there is no RCP, or if the RCP allows the requested action, the enforcement code evaluation continues.
- **AWS Organizations SCPs** – The enforcement code evaluates AWS Organizations service control policies (SCPs) that apply to the request. SCPs apply to principals of the account where the SCPs are attached. If the enforcement code does not find any applicable `Allow` statements in the SCPs, the enforcement code returns a final decision of **Deny**. If there is no SCP, or if the SCP allows the requested action, the enforcement code evaluation continues.
- **Resource-based policies** – Within the same account, resource-based policies impact policy evaluation differently depending on the type of principal accessing the resource, and the principal that is allowed in the resource-based policy. Depending on the type of principal, an `Allow` in a resource-based policy can result in a final decision of `Allow`, even if an implicit deny in an identity-based policy, permissions boundary, or session policy is present.

  For most resources, you only need an explicit `Allow` for the principal in either an identity-based policy or a resource-based policy to grant access. IAM role trust policies and KMS key policies are exceptions to this logic, because they must explicitly allow access for principals. Resource-based policies for services other than IAM and AWS KMS may also require an explicit `Allow` statement within the same account to grant access. For more information, see the documentation for the specific service you're working with.

  For single account policy evaluation requests, resource-based policy logic differs from other policy types if the specified principal is an IAM user, an IAM role, or a session principal. Session principals include IAM role sessions or an AWS STS federated user principals. If a resource-based policy grants permission directly to the IAM user or the session principal that is making the request, then an implicit deny in an identity-based policy, a permissions boundary, or a session policy does not impact the final decision.

  - **IAM role** – Resource-based policies that grant permissions to an IAM role ARN are limited by an implicit deny in a permissions boundary or session policy. You can specify the role ARN in the

Principal element or the `aws:PrincipalArn` condition key. In both cases, the principal that makes the request is the **IAM role session**.

Permissions boundaries and session policies do not limit permissions granted using the `aws:PrincipalArn` condition key with a wildcard(*) in the Principal element, unless the identity-based policies contain an explicit deny. For more information, see IAM role principals.

Example role ARN

```
arn:aws:iam::111122223333:role/examplerole
```

- **IAM role session** – Within the same account, resource-based policies that grant permissions to an IAM role session ARN grant permissions directly to the assumed role session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy. When you assume a role and make a request, the principal making the request is the IAM role session ARN and not the ARN of the role itself. For more information, see Role session principals.

  Example role session ARN

  ```
  arn:aws:sts::111122223333:assumed-role/examplerole/examplerolesessionname
  ```

- **IAM user** – Within the same account, resource-based policies that grant permissions to an IAM user ARN (that is not a federated user session) are not limited by an implicit deny in an identity-based policy or permissions boundary.

  Example IAM user ARN

  ```
  arn:aws:iam::111122223333:user/exampleuser
  ```

- **AWS STS federated user principal** – A federated user session is a session created by calling `GetFederationToken`. When a federated user makes a request, the principal making the request is the federated user ARN and not the ARN of the IAM user who federated. Within the same account, resource-based policies that grant permissions to a federated user ARN grant permissions directly to the session. Permissions granted directly to a session are not limited by an implicit deny in an identity-based policy, a permissions boundary, or session policy.

  However, if a resource-based policy grants permission to the ARN of the IAM user who federated, then requests made by the federated user during the session are limited by an implicit deny in a permission boundary or session policy.

Example federated user session ARN

```
arn:aws:sts::111122223333:federated-user/exampleuser
```

- **Identity-based policies** – The enforcement code checks the identity-based policies for the principal. For an IAM user, these include user policies and policies from groups to which the user belongs. If there are no identity-based policies or no statements in identity-based policies that allow the requested action, then the request is implicitly denied and the enforcement code returns a final decision of **Deny**. If any statement in any applicable identity-based policies allows the requested action, the code evaluation continues.
- **IAM permissions boundaries** – The enforcement code checks whether the IAM entity that is used by the principal has a permissions boundary. If the policy that is used to set the permissions boundary does not allow the requested action, then the request is implicitly denied. The enforcement code returns a final decision of **Deny**. If there is no permissions boundary, or if the permissions boundary allows the requested action, the code evaluation continues.
- **Session policies** – The enforcement code checks whether the principal is a session principal. Session principals include an IAM role session or an AWS STS federated user session. If the principal is not a session principal, the enforcement code returns a final decision of **Allow**.

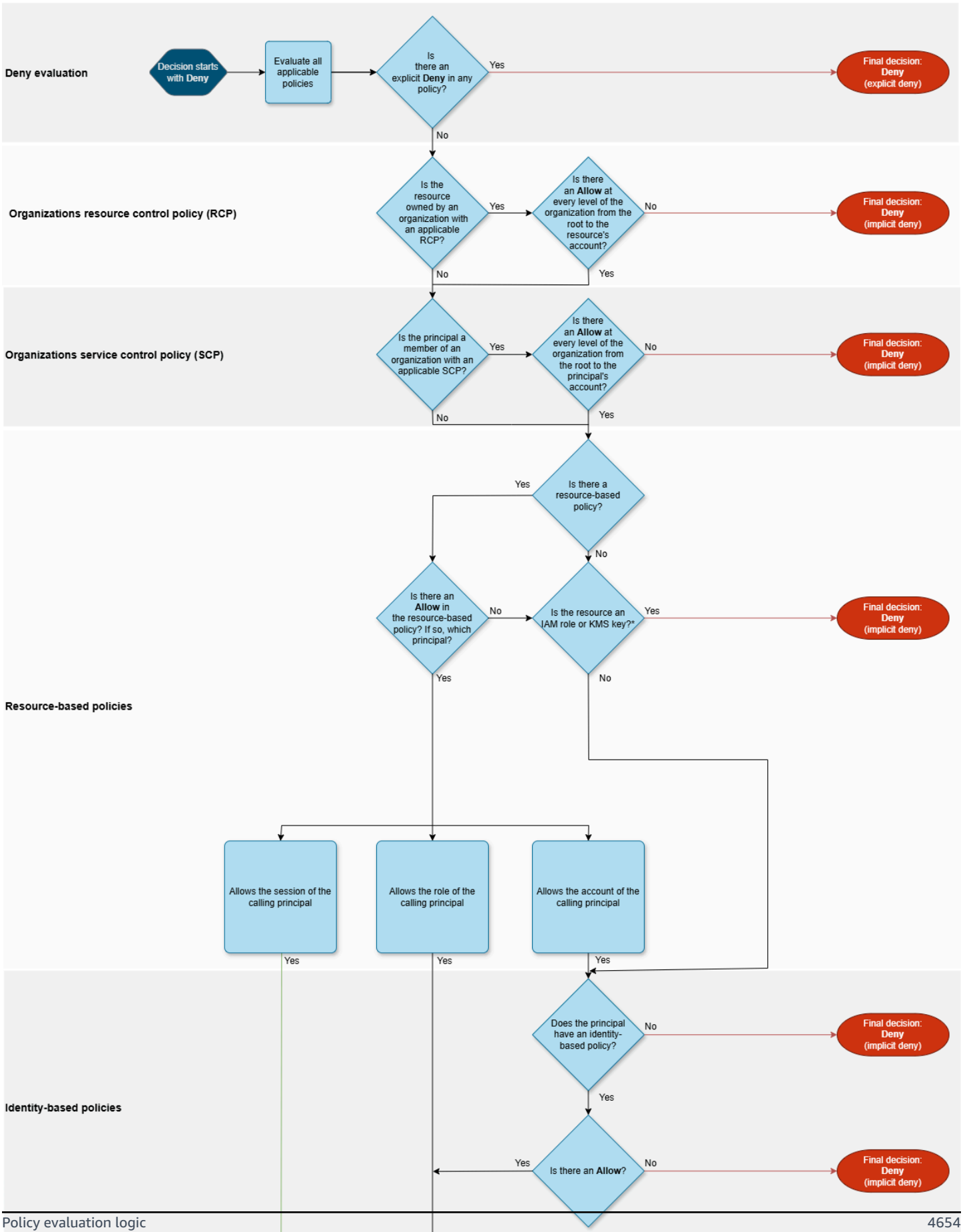
  For session principals, the enforcement code checks whether a session policy was passed in the request. You can pass a session policy while using the AWS CLI or AWS API to get temporary credentials for a role or an AWS STS federated user principal. If you didn't pass a session policy, a default session policy is created and the enforcement code returns a final decision of **Allow**.

  - If a session policy is present and does not allow the requested action, then the request is implicitly denied. The enforcement code returns a final decision of **Deny**.
  - The enforcement code checks whether the principal is a role session. If the principal is a role session, then the request is **Allowed**. Otherwise, the request is implicitly denied and the enforcement code returns a final decision of **Deny**.
  - If a session policy is present and allows the requested action, then the enforcement code returns a final decision of **Allow**.

Policy evaluation for requests within a single account

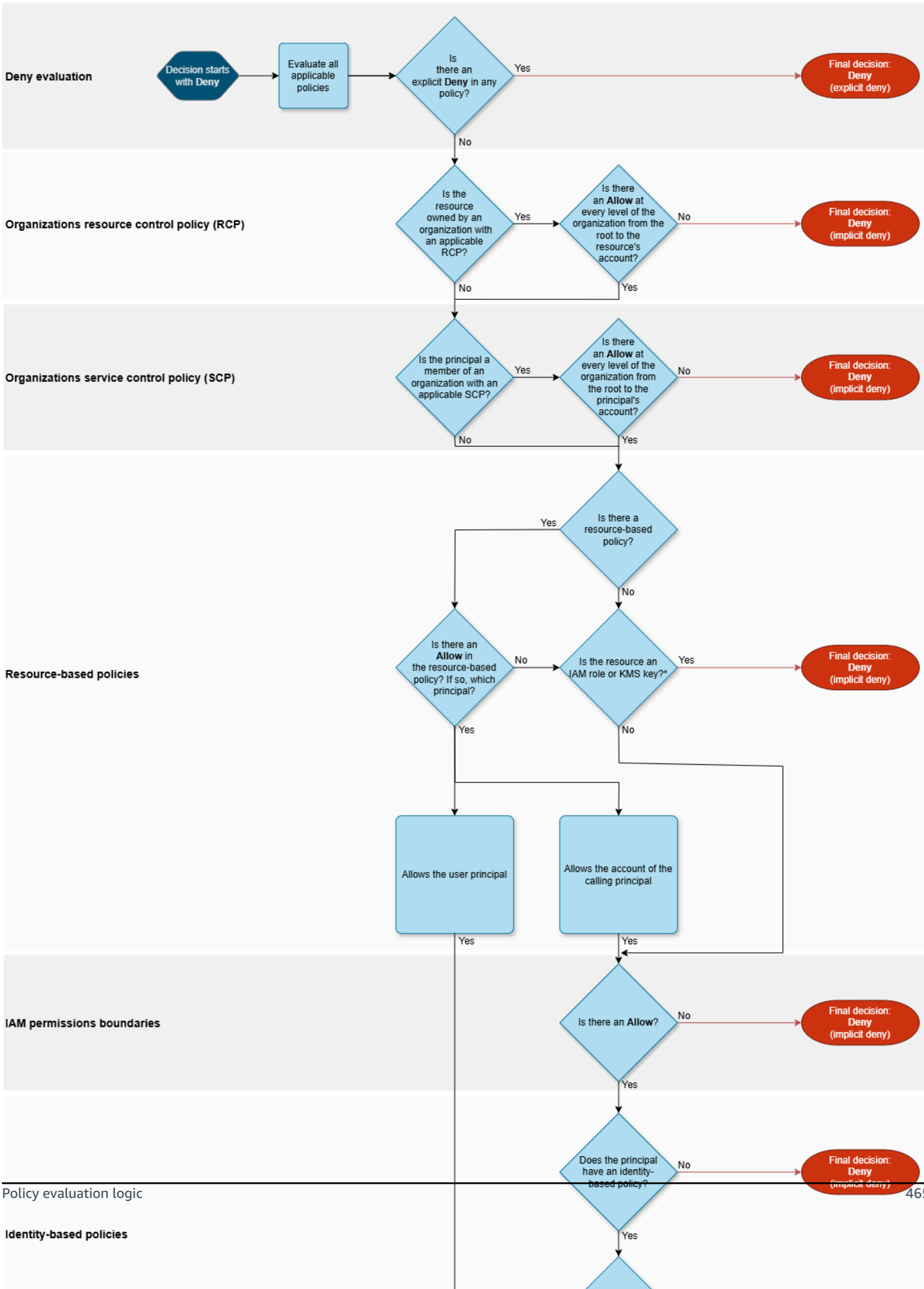
Policy evaluation for an IAM role

The following flow chart provides details about how a policy evaluation decision is made for an IAM role within a single account.


Deny evaluation
Decision starts with Deny
Evaluate all applicable policies
Is there an explicit Deny in any policy?
Yes
Final decision: Deny (explicit deny)
No
Organizations resource control policy (RCP)
Is the resource owned by an organization with an applicable RCP?
Yes
Is there an Allow at every level of the organization from the root to the resource's account?
No
Final decision: Deny (implicit deny)
No
Yes
Organizations service control policy (SCP)
Is the principal a member of an organization with an applicable SCP?
Yes
Is there an Allow at every level of the organization from the root to the principal's account?
No
Final decision: Deny (implicit deny)
No
Yes
Resource-based policies
Yes
Is there a resource-based policy?
No
Is there an Allow in the resource-based policy? If so, which principal?
No
Is the resource an IAM role or KMS key?*
Yes
Final decision: Deny (implicit deny)
Yes
No
Allows the session of the calling principal
Allows the role of the calling principal
Allows the account of the calling principal
Yes
Yes
Yes
Identity-based policies
Does the principal have an identity-based policy?
No
Final decision: Deny (implicit deny)
Yes
Yes
Is there an Allow?
No
Final decision: Deny (implicit deny)
IAM permissions boundaries
Does the principal have a permissions
Yes
Is there an Allow?
No
Final decision: Deny

Policy evaluation for an IAM user

The following flow chart provides details about how a policy evaluation decision is made for an IAM user within a single account.


Deny evaluation
Decision starts with Deny
Evaluate all applicable policies
Is there an explicit Deny in any policy?
Yes
Final decision: Deny (explicit deny)
No
Organizations resource control policy (RCP)
Is the resource owned by an organization with an applicable RCP?
Yes
Is there an Allow at every level of the organization from the root to the resource's account?
No
Final decision: Deny (implicit deny)
No
Yes
Organizations service control policy (SCP)
Is the principal a member of an organization with an applicable SCP?
Yes
Is there an Allow at every level of the organization from the root to the principal's account?
No
Final decision: Deny (implicit deny)
No
Yes
Resource-based policies
Is there a resource-based policy?
Yes
No
Is there an Allow in the resource-based policy? If so, which principal?
No
Is the resource an IAM role or KMS key?*
Yes
Final decision: Deny (implicit deny)
Yes
No
Allows the user principal
Allows the account of the calling principal
Yes
IAM permissions boundaries
Is there an Allow?
No
Final decision: Deny (implicit deny)
Yes
Identity-based policies
Does the principal have an identity-based policy?
No
Final decision: Deny (implicit deny)
Yes

Example identity-based and resource-based policy evaluation

The most common types of policies are identity-based policies and resource-based policies. When access to a resource is requested, AWS evaluates all the permissions granted by the policies for **at least one Allow** within the same account. An explicit deny in any of the policies overrides the allow.

> **Important**
>
> If either the identity-based policy or the resource-based policy within the same account allows the request and the other doesn't, the request is still allowed.

Assume that Carlos has the user name `carlossalazar` and he tries to save a file to the `amzn-s3-demo-bucket-carlossalazar-logs` Amazon S3 bucket.

Also assume that the following policy is attached to the `carlossalazar` IAM user.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3ListRead",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAccessPoints",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "AllowS3Self",
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket-carlossalazar/*",
                "arn:aws:s3:::amzn-s3-demo-bucket-carlossalazar"
            ]
```

```
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::*log*"
        }
    ]
}
```

The `AllowS3ListRead` statement in this policy allows Carlos to view a list of all of the buckets in the account. The `AllowS3Self` statement allows Carlos full access to the bucket with the same name as his user name. The `DenyS3Logs` statement denies Carlos access to any S3 bucket with `log` in its name.

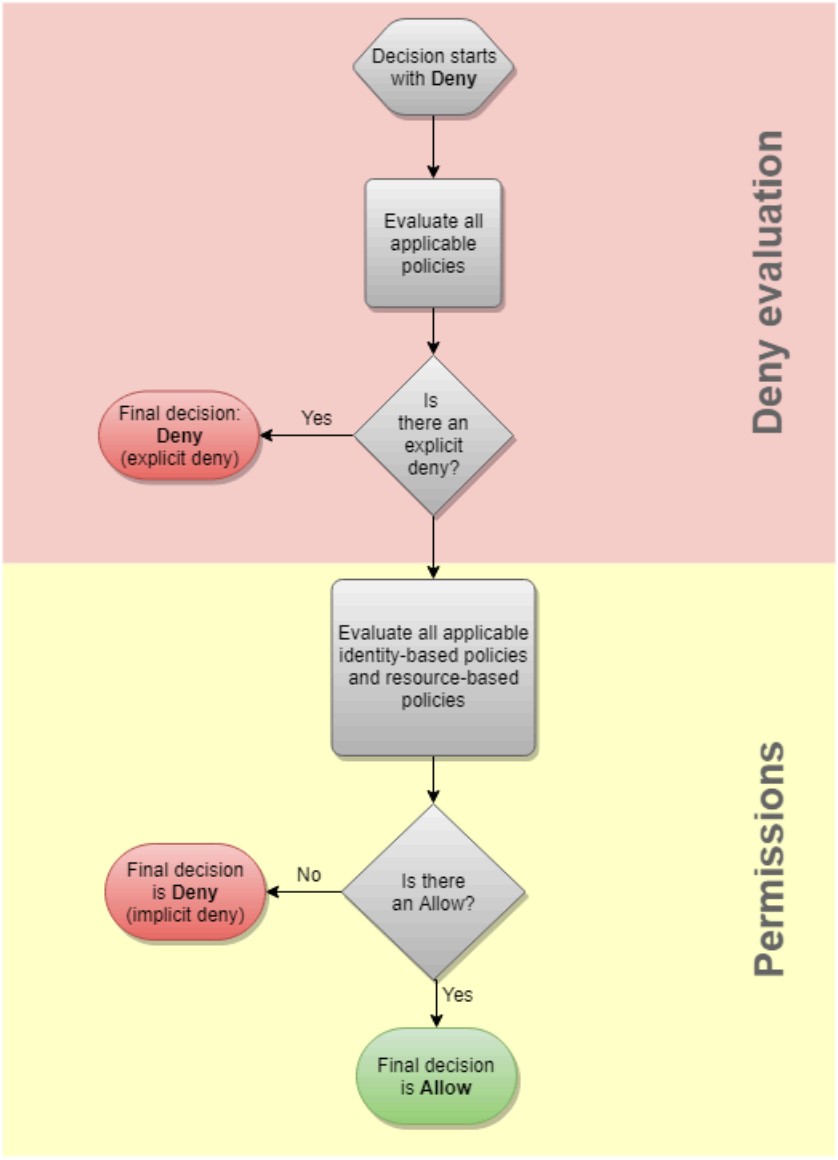
Additionally, the following resource-based policy (called a bucket policy) is attached to the `amzn-s3-demo-bucket-carlossalazar` bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/carlossalazar"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket-carlossalazar/*",
                "arn:aws:s3:::amzn-s3-demo-bucket-carlossalazar"
            ]
        }
    ]
}
```

This policy specifies that only the `carlossalazar` user can access the `amzn-s3-demo-bucket-carlossalazar` bucket.

When Carlos makes his request to save a file to the `amzn-s3-demo-bucket-carlossalazar-logs` bucket, AWS determines what policies apply to the request. In this case, only the identity-based policy and the resource-based policy apply. These are both permissions policies. Because no permissions boundaries apply, the evaluation logic is reduced to the following logic.



AWS first checks for a `Deny` statement that applies to the context of the request. It finds one, because the identity-based policy explicitly denies Carlos access to any S3 buckets used for logging. Carlos is denied access.

Assume that he then realizes his mistake and tries to save the file to the `amzn-s3-demo-bucket-carlossalazar` bucket. AWS checks for a `Deny` statement and does not find one. It then checks the permissions policies. Both the identity-based policy and the resource-based policy allow the request. Therefore, AWS allows the request. If either of them explicitly denied the statement,

the request would have been denied. If one of the policy types allows the request and the other doesn't, the request is still allowed.

Cross-account policy evaluation logic

You can allow a principal in one account to access resources in a second account. This is called cross-account access. When you allow cross-account access, the account where the principal exists is called the *trusted* account. The account where the resource exists is the *trusting* account.

To allow cross-account access, you attach a resource-based policy to the resource that you want to share. You must also attach an identity-based policy to the identity that acts as the principal in the request. The resource-based policy in the trusting account must specify the principal of the trusted account that will have access to the resource. You can specify the entire account or its IAM users, AWS STS federated user principals, IAM roles, or assumed-role sessions. You can also specify an AWS service as a principal. For more information, see How to specify a principal.

The principal's identity-based policy must allow the requested access to the resource in the trusting service. You can do this by specifying the ARN of the resource.

In IAM, you can attach a resource-based policy to an IAM role to allow principals in other accounts to assume that role. The role's resource-based policy is called a role trust policy. After assuming that role, the allowed principals can use the resulting temporary credentials to access multiple resources in your account. This access is defined in the role's identity-based permissions policy. To learn how allowing cross-account access using roles is different from allowing cross-account access using other resource-based policies, see Cross account resource access in IAM.

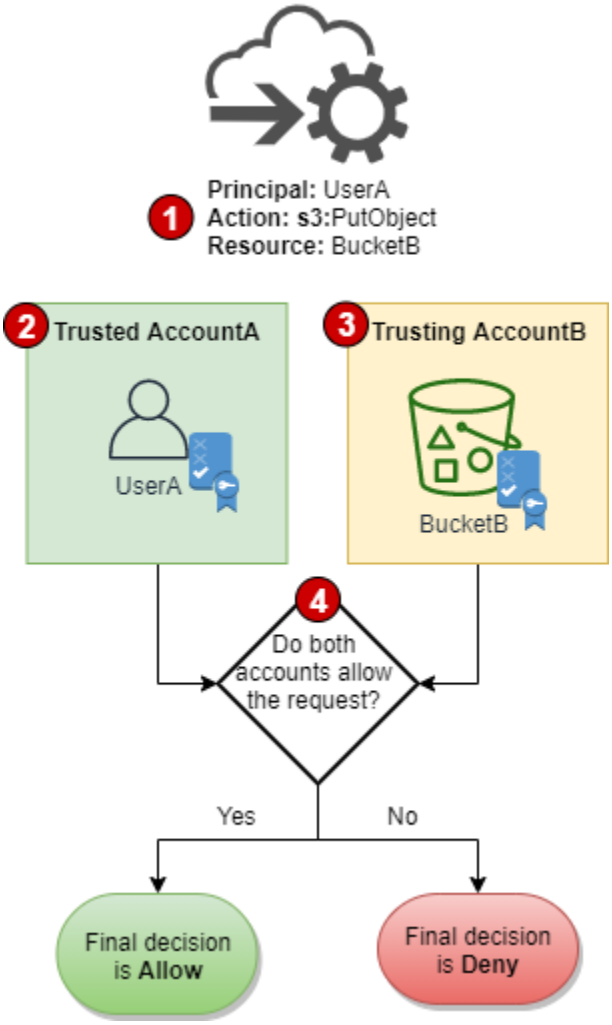
> **Important**
>
> Other services can affect the policy evaluation logic. For example, AWS Organizations supports service control policies and resource control policies that can be applied to principals and resources in one or more accounts. AWS Resource Access Manager supports policy fragments that control which actions that principals are allowed to perform on resources that are shared with them.

Determining whether a cross-account request is allowed

For cross-account requests, the requester in the trusted `AccountA` must have an identity-based policy. That policy must allow them to make a request to the resource in the trusting `AccountB`.

Additionally, the resource-based policy in `AccountB` must allow the requester in `AccountA` to access the resource.

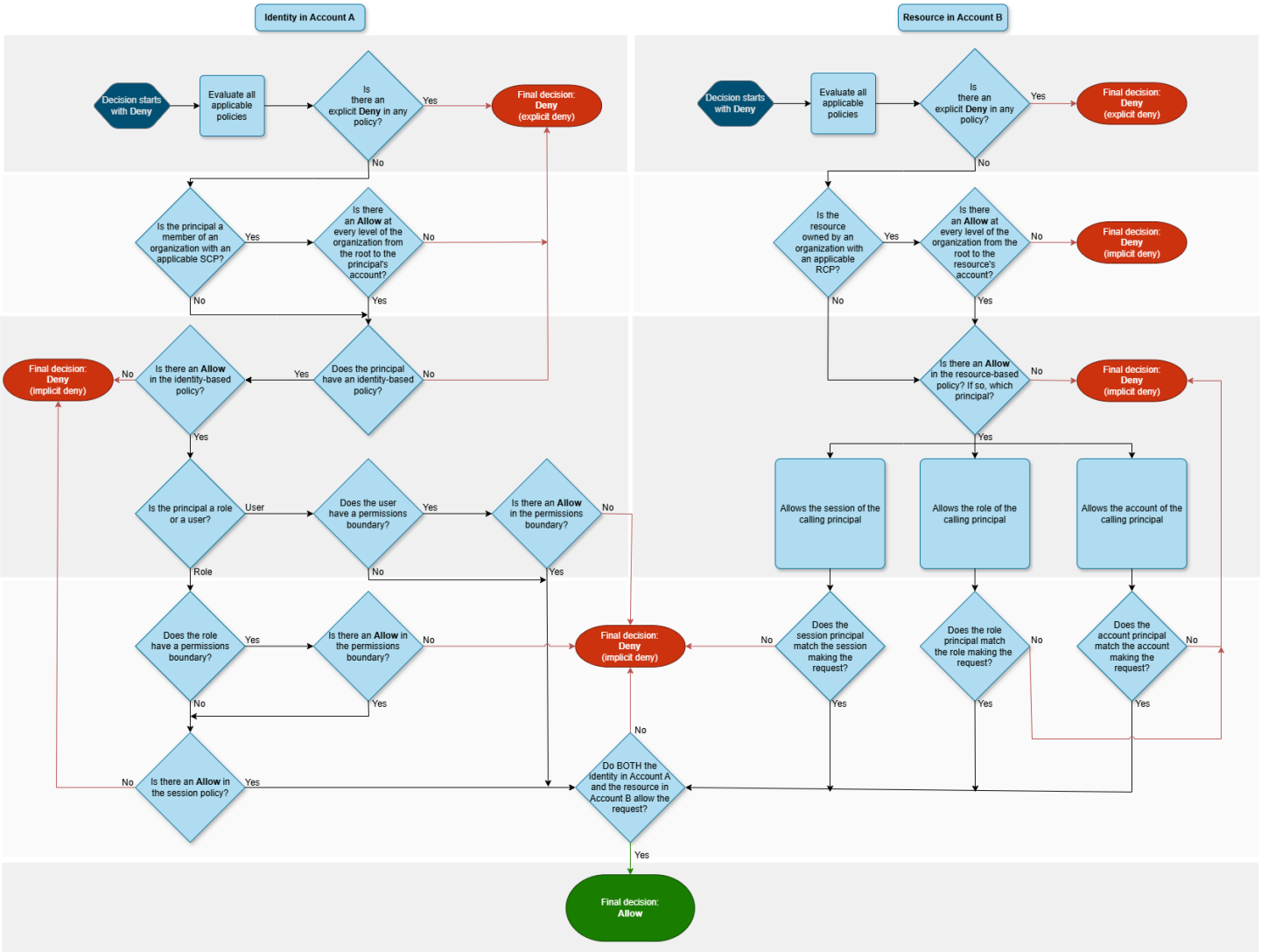
When you make a cross-account request, AWS performs two evaluations. AWS evaluates the request in the trusting account and the trusted account. For more information about how a request is evaluated within a single account, see How AWS enforcement code logic evaluates requests to allow or deny access. The request is allowed only if both evaluations return a decision of `Allow`.



1. When a principal in one account makes a request to access a resource in another account, this is a cross-account request.
2. The requesting principal exists in the trusted account (`AccountA`). When AWS evaluates this account, it checks the identity-based policy and any policies that can limit an identity-based policy. For more information, see Evaluating identity-based policies with permissions boundaries.

3. The requested resource exists in the trusting account (`AccountB`). When AWS evaluates this account, it checks the resource-based policy that is attached to the requested resource and any policies that can limit a resource-based policy. For more information, see [Evaluating identity-based policies with resource-based policies].
4. AWS allows the request only if both account policy evaluations allow the request.

The following flow chart provides a more detailed illustration of how a policy evaluation decision is made for a cross-account request. Again, AWS allows the request only if both account policy evaluations allow the request.



Example cross-account policy evaluation

The following example demonstrates a scenario where a role in one account is granted permissions by a resource-based policy in a second account.

Assume that Carlos is a developer with an IAM role named Demo in account 111111111111. He wants to save a file to the amzn-s3-demo-bucket-production-logs Amazon S3 bucket in account 222222222222.

Also assume that the following policy is attached to the Demo IAM role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3ListRead",
            "Effect": "Allow",
            "Action": "s3:ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Sid": "AllowS3ProductionObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object*",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket-production/*"
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::*log*",
                "arn:aws:s3:::*log*/*"
            ]
        }
    ]
}
```

The AllowS3ListRead statement in this policy allows Carlos to view a list of all of the buckets in Amazon S3. The AllowS3ProductionObjectActions statement allows Carlos full access to objects in the amzn-s3-demo-bucket-production bucket.

Additionally, the following resource-based policy (called a bucket policy) is attached to the amzn-s3-demo-bucket-production bucket in account 222222222222.

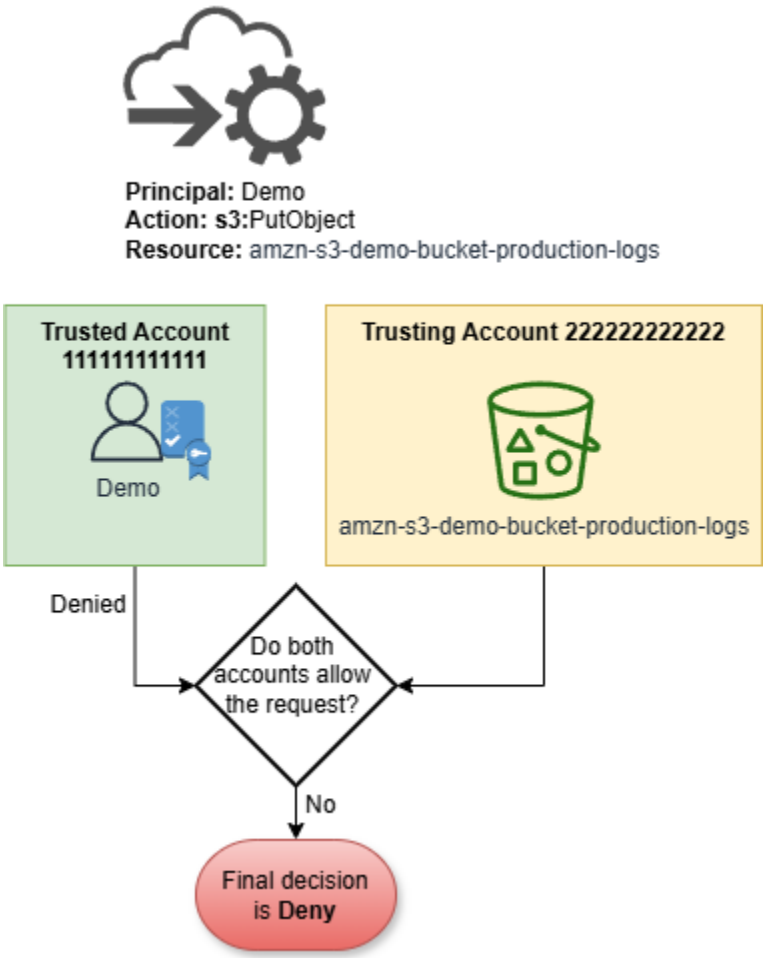
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:PutObject*",
                "s3:ReplicateObject",
                "s3:RestoreObject"
            ],
            "Principal": { "AWS": "arn:aws:iam::111111111111:role/Demo" },
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket-production/*"
        }
    ]
}
```

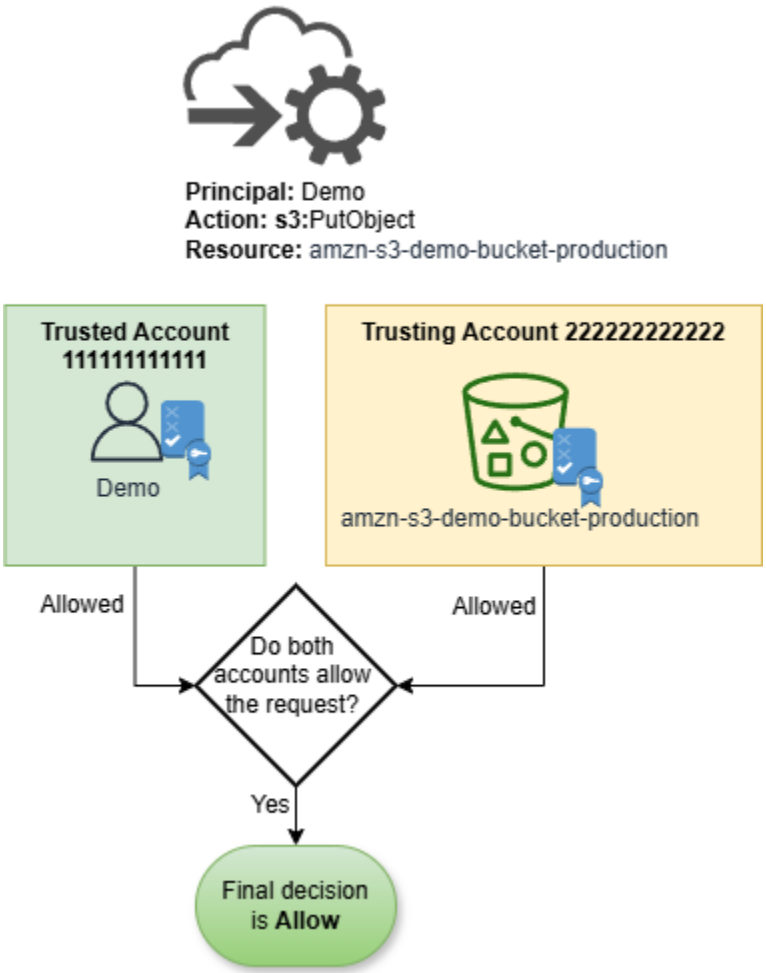
This policy allows the Demo role to access objects in the amzn-s3-demo-bucket-production bucket. The role can create and edit, but not delete the objects in the bucket. The role can't manage the bucket itself.

When Carlos makes his request to save a file to the amzn-s3-demo-bucket-production-logs bucket, AWS determines what policies apply to the request. In this case, the identity-based policy attached to the Demo role is the only policy that applies in account 111111111111. In account 222222222222, there is no resource-based policy attached to the amzn-s3-demo-bucket-production-logs bucket. When AWS evaluates account 111111111111, it returns a decision of Deny. This is because the DenyS3Logs statement in the identity-based policy explicitly denies access to any log buckets. For more information about how a request is evaluated within a single account, see How AWS enforcement code logic evaluates requests to allow or deny access.

Because the request is explicitly denied within one of the accounts, the final decision is to deny the request.



Assume that Carlos then realizes his mistake and tries to save the file to the `Production` bucket. AWS first checks account 111111111111 to determine if the request is allowed. Only the identity-based policy applies, and it allows the request. AWS then checks account 222222222222. Only the resource-based policy attached to the `Production` bucket applies, and it allows the request. Because both accounts allow the request, the final decision is to allow the request.



The difference between explicit and implicit denies

A request results in an explicit deny if an applicable policy includes a `Deny` statement. If policies that apply to a request include an `Allow` statement and a `Deny` statement, the `Deny` statement trumps the `Allow` statement. The request is explicitly denied.

An implicit denial occurs when there is no applicable `Deny` statement but also no applicable `Allow` statement. Because an IAM principal is denied access by default, they must be explicitly allowed to perform an action. Otherwise, they are implicitly denied access.

When you design your authorization strategy, you must create policies with `Allow` statements to allow your principals to successfully make requests. However, you can choose any combination of explicit and implicit denies.

For example, you can create the following policy that includes allowed actions, implicitly denied actions, and explicitly denied actions. The `AllowGetList` statement **allows** read-only access

to IAM actions that begin with the prefixes `Get` and `List`. All other actions in IAM, such as `iam:CreatePolicy` are **implicitly denied**. The `DenyReports` statement **explicitly denies** access to IAM reports by denying access to actions that include the `Report` suffix, such as `iam:GetOrganizationsAccessReport`. If someone adds another policy to this principal to grant them access to IAM reports, such as `iam:GenerateCredentialReport`, report-related requests are still denied because of this explicit deny.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGetList",
            "Effect": "Allow",
            "Action": [
                "iam:Get*",
                "iam:List*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DenyReports",
            "Effect": "Deny",
            "Action": "iam:*Report",
            "Resource": "*"
        }
    ]
}
```

Grammar of the IAM JSON policy language

This page presents a formal grammar for the language used to create JSON policies in IAM. We present this grammar so that you can understand how to construct and validate policies.

For examples of policies, see the following topics:

- Policies and permissions in AWS Identity and Access Management
- Example IAM identity-based policies

- Example Policies for Working in the Amazon EC2 Console and Example Policies for Working With the AWS CLI, the Amazon EC2 CLI, or an AWS SDK in the *Amazon EC2 User Guide*.
- Bucket Policy Examples and User Policy Examples in the *Amazon Simple Storage Service User Guide*.

For examples of policies used in other AWS services, go to the documentation for those services.

Topics

- The policy language and JSON
- Conventions used in this grammar
- Grammar
- Policy grammar notes

The policy language and JSON

Policies are expressed in JSON. When you create or edit a JSON policy, IAM can perform policy validation to help you create an effective policy. IAM identifies JSON syntax errors, while IAM Access Analyzer provides additional policy checks with recommendations to help you further refine your policies. To learn more about policy validation, see IAM policy validation. To learn more about IAM Access Analyzer policy checks and actionable recommendations, see IAM Access Analyzer policy validation.

In this document, we do not provide a complete description of what constitutes valid JSON. However, here are some basic JSON rules:

- White space between individual entities is allowed.
- Values are enclosed in quotation marks. Quotation marks are optional for numeric and Boolean values.
- Many elements (for example, `action_string_list` and `resource_string_list`) can take a JSON array as a value. Arrays can take one or more values. If more than one value is included, the array is in square brackets ([and]) and comma-delimited, as in the following example:

```
"Action" : ["ec2:Describe*","ec2:List*"]
```

- Basic JSON data types (Boolean, number, and string) are defined in RFC 7159.

Conventions used in this grammar

The following conventions are used in this grammar:

- The following characters are JSON tokens and *are* included in policies:

  ```
  { } [ ] " , :
  ```

- The following characters are special characters in the grammar and are *not* included in policies:

  ```
  = < > ( ) |
  ```

- If an element allows multiple values, it is indicated using repeated values, a comma delimiter, and an ellipsis (`...`). Examples:

  ```
  [<action_string>, <action_string>, ...]
  ```

  ```
  <principal_map> = { <principal_map_entry>, <principal_map_entry>, ... }
  ```

  If multiple values are allowed, it is also valid to include only one value. For only one value, the trailing comma must be omitted. If the element takes an array (marked with [and]) but only one value is included, the brackets are optional. Examples:

  ```
  "Action": [<action_string>]
  ```

  ```
  "Action": <action_string>
  ```

- A question mark (?) following an element indicates that the element is optional. Example:

  ```
  <version_block?>
  ```

  However, be sure to refer to the notes that follow the grammar listing for details about optional elements.

- A vertical line (`|`) between elements indicates alternatives. In the grammar, parentheses define the scope of the alternatives. Example:

  ```
  ("Principal" | "NotPrincipal")
  ```

- Elements that must be literal strings are enclosed in double quotation marks ("). Example:

  ```
  <version_block> = "Version" : ("2008-10-17" | "2012-10-17" )
  ```

For additional notes, see Policy grammar notes following the grammar listing.

Grammar

The following listing describes the policy language grammar. For conventions used in the listing, see the preceding section. For additional information, see the notes that follow.

> **Note**
>
> This grammar describes policies marked with a version of `2008-10-17` and `2012-10-17`. A `Version` policy element is different from a policy version. The `Version` policy element is used within a policy and defines the version of the policy language. A policy version, on the other hand, is created when you make changes to a customer managed policy in IAM. The changed policy doesn't overwrite the existing policy. Instead, IAM creates a new version of the managed policy. To learn more about the `Version` policy element see IAM JSON policy elements: Version. To learn more about policy versions, see the section called "Versioning IAM policies".

```
policy  = {
     <version_block?>,
     <id_block?>,
     <statement_block>
}

<version_block> = "Version" : ("2008-10-17"        | "2012-10-17"       )

<id_block> = "Id" : <policy_id_string>

<statement_block> = "Statement" : [ <statement>, <statement>, ... ]

<statement> = {
    <sid_block?>,
    <principal_block?>,
    <effect_block>,
    <action_block>,
    <resource_block>,
    <condition_block?>
}

<sid_block> = "Sid" : <sid_string>

<effect_block> = "Effect" : ("Allow" | "Deny")
```

```
<principal_block> = ("Principal" | "NotPrincipal") : ("*" | <principal_map>)

<principal_map> = { <principal_map_entry>, <principal_map_entry>, ... }

<principal_map_entry> = ("AWS" | "Federated" | "Service" | "CanonicalUser") :
    [<principal_id_string>, <principal_id_string>, ...]

<action_block> = ("Action" | "NotAction") :
    ("*" | <action_string> | [<action_string>, <action_string>, ...])

<resource_block> = ("Resource" | "NotResource") :
    : ("*" | <resource_string> | [<resource_string>, <resource_string>, ...])

<condition_block> = "Condition" : { <condition_map> }
<condition_map> = {
  <condition_type_string> : { <condition_key_string> : <condition_value_list> },
  <condition_type_string> : { <condition_key_string> : <condition_value_list> }, ...
}
<condition_value_list> = [<condition_value>, <condition_value>, ...]
<condition_value> = (<condition_value_string> | <condition_value_string> |
 <condition_value_string>)
```

Policy grammar notes

- A single policy can contain an array of statements.
- Policies have a maximum size between 2048 characters and 10,240 characters, depending on what entity the policy is attached to. For more information, see IAM and AWS STS quotas. Policy size calculations do not include white space characters.
- Individual elements must not contain multiple instances of the same key. For example, you cannot include the `Effect` block twice in the same statement.
- Blocks can appear in any order. For example, `version_block` can follow `id_block` in a policy. Similarly, `effect_block`, `principal_block`, `action_block` can appear in any order within a statement.
- The `id_block` is optional in resource-based policies. It must *not* be included in identity-based policies.
- The `principal_block` element is required in resource-based policies (for example, in Amazon S3 bucket policies) and in trust policies for IAM roles. It must *not* be included in identity-based policies.

- The `principal_map` element in Amazon S3 bucket policies can include the `CanonicalUser` ID. Most resource-based policies do not support this mapping. To learn more about using the canonical user ID in a bucket policy, see Specifying a Principal in a Policy in the *Amazon Simple Storage Service User Guide*.
- Each string value (`policy_id_string`, `sid_string`, `principal_id_string`, `action_string`, `resource_string`, `condition_type_string`, `condition_key_string`, and the string version of `condition_value`) can have its own minimum and maximum length restrictions, specific allowed values, or required internal format.

Notes about string values

This section provides additional information about string values that are used in different elements in a policy.

`action_string`

Consists of a service namespace, a colon, and the name of an action. Action names can include wildcards. Examples:

```
"Action":"ec2:StartInstances"

"Action":[
  "ec2:StartInstances",
  "ec2:StopInstances"
]

"Action":"cloudformation:*"

"Action":"*"

"Action":[
  "s3:Get*",
  "s3:List*"
]
```

`policy_id_string`

Provides a way to include information about the policy as a whole. Some services, such as Amazon SQS and Amazon SNS, use the `Id` element in reserved ways. Unless otherwise

restricted by an individual service, policy_id_string can include spaces. Some services require this value to be unique within an AWS account.

> **Note**
>
> The `id_block` is allowed in resource-based policies, but not in identity-based policies.

There is no limit to the length, although this string contributes to the overall length of the policy, which is limited.

```
"Id":"Admin_Policy"

"Id":"cd3ad3d9-2776-4ef1-a904-4c229d1642ee"
```

`sid_string`

Provides a way to include information about an individual statement. For IAM policies, basic alphanumeric characters (A-Z,a-z,0-9) are the only allowed characters in the `Sid` value. Other AWS services that support resource policies may have other requirements for the `Sid` value. For example, some services require this value to be unique within an AWS account, and some services allow additional characters such as spaces in the `Sid` value.

```
"Sid":"1"

"Sid": "ThisStatementProvidesPermissionsForConsoleAccess"
```

`principal_id_string`

Provides a way to specify a principal using the *Amazon Resource Name* (ARN) of the AWS account, IAM user, IAM role, federated user, or assumed-role user. For an AWS account, you can also use the short form `AWS:accountnumber` instead of the full ARN. For all of the options including AWS services, assumed roles, and so on, see How to specify a principal.

Note that you can use * only to specify "everyone/anonymous." You cannot use it to specify part of a name or ARN.

`resource_string`

In most cases, consists of an Amazon Resource Name (ARN). You can use wildcards (* and ?) in the resource portion of the ARN. For more information about using wildcards in ARNs, see Using wildcards in paths.

> **Note**
>
> When you specify an incomplete ARN (one with fewer than the standard six fields) in an identity-based policy, AWS automatically completes the ARN by adding wildcard characters (*) to all missing fields. For example, specifying `arn:aws:sqs` is equivalent to `arn:aws:sqs:*:*:*`, which grants access to all Amazon Amazon SQS resources across all regions and accounts. However, session policies passed to AWS STS AssumeRole, AssumeRoleWithWebIdentity, and AssumeRoleWithSAML requests, do not support incomplete ARNs. Using an incomplete ARN in a session policy will result in a `MalformedPolicyDocumentException` error.

```
"Resource":"arn:aws:iam::123456789012:user/Bob"

"Resource":"arn:aws:s3:::amzn-s3-demo-bucket/*"
```

`condition_type_string`

Identifies the type of condition being tested, such as `StringEquals`, `StringLike`, `NumericLessThan`, `DateGreaterThanEquals`, `Bool`, `BinaryEquals`, `IpAddress`, `ArnEquals`, etc. For a complete list of condition types, see IAM JSON policy elements: Condition operators.

```
"Condition": {
  "NumericLessThanEquals": {
    "s3:max-keys": "10"
  }
}

"Condition": {
  "Bool": {
    "aws:SecureTransport": "true"
  }
}
```

```
"Condition": {
  "StringEquals": {
      "s3:x-amz-server-side-encryption": "AES256"
   }
}
```

condition_key_string

Identifies the condition key whose value will be tested to determine whether the condition is met. AWS defines a set of condition keys that are available in all AWS services, including `aws:PrincipalType`, `aws:SecureTransport`, and `aws:userid`.

For a list of AWS condition keys, see AWS global condition context keys. For condition keys that are specific to a service, see the documentation for that service such as the following:

- Specifying Conditions in a Policy in the *Amazon Simple Storage Service User Guide*
- IAM Policies for Amazon EC2 in the *Amazon EC2 User Guide*.

```
"Condition":{
  "Bool": {
      "aws:SecureTransport": "true"
   }
}

"Condition": {
  "StringNotEquals": {
      "s3:x-amz-server-side-encryption": "AES256"
   }
}

"Condition": {
  "StringEquals": {
    "aws:ResourceTag/purpose": "test"
  }
}
```

condition_value_string

Identifies the value of the condition_key_string that determines whether the condition is met. For a complete list of valid values for a condition type, see IAM JSON policy elements: Condition operators.

```
"Condition":{
  "ForAnyValue:StringEquals": {
  "dynamodb:Attributes": [
   "ID",
   "PostDateTime"
         ]
  }
}
```

AWS managed policies for job functions

We recommend using policies that grant least privilege, or granting only the permissions required to perform a task. The most secure way to grant least privilege is to write a custom policy with only the permissions needed by your team. You must create a process to allow your team to request more permissions when necessary. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need.

To get started adding permissions to your IAM identities (users, groups of users, and roles), you can use AWS managed policies. AWS managed policies cover common use cases and are available in your AWS account. AWS managed policies don't grant least privilege permissions. You must consider the security risk of granting your principals more permissions than they need to do their job.

You can attach AWS managed policies, including job functions, to any IAM identity. To switch to least privilege permissions, you can run AWS Identity and Access Management and Access Analyzer to monitor principals with AWS managed policies. After learning which permissions they are using, then you can write a custom policy or generate a policy with only the required permissions for your team. This is less secure, but provides more flexibility as you learn how your team is using AWS.

AWS managed policies for job functions are designed to closely align to common job functions in the IT industry. You can use these policies to grant the permissions needed to carry out the tasks expected of someone in a specific job function. These policies consolidate permissions for many services into a single policy that's easier to work with than having permissions scattered across many policies.

Use Roles to Combine Services

Some of the policies use IAM service roles to help you take advantage of features found in other AWS services. These policies grant access to `iam:passrole`, which allows a user with the policy to

pass a role to an AWS service. This role delegates IAM permissions to the AWS service to carry out actions on your behalf.

You must create the roles according to your needs. For example, the Network Administrator policy allows a user with the policy to pass a role named "flow-logs-vpc" to the Amazon CloudWatch service. CloudWatch uses that role to log and capture IP traffic for VPCs created by the user.

To follow security best practices, the policies for job functions include filters that limit the names of valid roles that can be passed. This helps avoid granting unnecessary permissions. If your users do require the optional service roles, you must create a role that follows the naming convention specified in the policy. You then grant permissions to the role. Once that is done, the user can configure the service to use the role, granting it whatever permissions the role provides.

In the following sections, each policy's name is a link to the policy details page in the AWS Management Console. There you can see the policy document and review the permissions it grants.

Administrator job function

AWS managed policy name: AdministratorAccess

Use case: This user has full access and can delegate permissions to every service and resource in AWS.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants all actions for all AWS services and for all resources in the account. For more information about the managed policy, see AdministratorAccess in *AWS Managed Policy Reference Guide*.

> **Note**
>
> Before an IAM user or role can access the AWS Billing and Cost Management console with the permissions in this policy, you must first activate IAM user and role access. To do this, follow the instructions in Grant access to the billing console to delegate access to the billing console.

Billing job function

AWS managed policy name: Billing

Use case: This user needs to view billing information, set up payments, and authorize payments. The user can monitor the costs accumulated for the entire AWS service.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants full permissions for managing billing, costs, payment methods, budgets, and reports. For additional cost management policy examples, see AWS Billing policy examples in the *AWS Billing and Cost Management User Guide*. For more information about the managed policy, see Billing in *AWS Managed Policy Reference Guide*.

> **Note**
>
> Before an IAM user or role can access the AWS Billing and Cost Management console with the permissions in this policy, you must first activate IAM user and role access. To do this, follow the instructions in Grant access to the billing console to delegate access to the billing console.

Database administrator job function

AWS managed policy name: DatabaseAdministrator

Use case: This user sets up, configures, and maintains databases in the AWS Cloud.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to create, configure, and maintain databases. It includes access to AWS database services, such as Amazon DynamoDB, Amazon Relational Database Service (RDS), and Amazon Redshift. View the policy for the full list of database services that this policy supports. For more information about the managed policy, see DatabaseAdministrator in *AWS Managed Policy Reference Guide*.

This job function policy supports the ability to pass roles to AWS services. The policy allows the `iam:PassRole` action for only those roles named in the following table. For more information, see Creating roles and attaching policies (console) later in this topic.

Use case	Role name (* is a wildcard)	Service role type to select	Select this AWS managed policy
Allow the user to monitor RDS databases	rds-monitoring-role	Amazon RDS Role for Enhanced Monitoring	AmazonRDSEnhancedMonitoringRole
Allow AWS Lambda to monitor your database and access external databases	rdbms-lambda-access	Amazon EC2	AWSLambda_FullAccess
Allow Lambda to upload files to Amazon S3 and to Amazon Redshift clusters with DynamoDB	lambda_exec_role	AWS Lambda	Create a new managed policy as defined in the AWS Big Data Blog
Allow Lambda functions to act as triggers for your DynamoDB tables	lambda-dynamodb-*	AWS Lambda	AWSLambda DynamoDBExecutionRole
Allow Lambda functions to access Amazon RDS in a VPC	lambda-vpc-execution-role	Create a role with a trust policy as defined in the AWS Lambda Developer Guide	AWSLambdaVPCAccessExecutionRole
Allow AWS Data Pipeline to access your AWS resources	DataPipelineDefaultRole	Create a role with a trust policy as defined in the AWS	The AWS Data Pipeline documentation lists the required permissions for this use case. See

Use case	Role name (* is a wildcard)	Service role type to select	Select this AWS managed policy
		Data Pipeline Developer Guide	IAM roles for AWS Data Pipeline
Allow your applications running on Amazon EC2 instances to access your AWS resources	DataPipelineDefaultResourceRole	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AmazonEC2RoleforDataPipelineRole

Data scientist job function

AWS managed policy name: DataScientist

Use case: This user runs Hadoop jobs and queries. The user also accesses and analyzes information for data analytics and business intelligence.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to create, manage, and run queries on an Amazon EMR cluster and perform data analytics with tools such as Amazon QuickSight. The policy includes access to additional data scientist services, such as AWS Data Pipeline, Amazon EC2, Amazon Kinesis, Amazon Machine Learning, and SageMaker AI. View the policy for the full list of data scientist services that this policy supports. For more information about the managed policy, see DataScientist in *AWS Managed Policy Reference Guide*.

This job function policy supports the ability to pass roles to AWS services. One statement allows passing any role to SageMaker AI. Another statement allows the `iam:PassRole` action for only those roles named in the following table. For more information, see Creating roles and attaching policies (console) later in this topic.

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
Allow Amazon EC2 instances access to services and resources suitable for clusters	EMR-EC2_DefaultRole	Amazon EMR for EC2	AmazonElasticMapReduceforEC2Role
Allow Amazon EMR access to access the Amazon EC2 service and resources for clusters	EMR_DefaultRole	Amazon EMR	AmazonEMRServicePolicy_v2
Allow Kinesis Managed Service for Apache Flink to access streaming data sources	kinesis-*	Create a role with a trust policy as defined in the AWS Big Data Blog.	See the AWS Big Data Blog, which outlines four possible options depending on your use case
Allow AWS Data Pipeline to access your AWS resources	DataPipelineDefaultRole	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	The AWS Data Pipeline documentation lists the required permissions for this use case. See IAM roles for AWS Data Pipeline
Allow your applications running on Amazon EC2 instances to access your AWS resources	DataPipelineDefaultResourceRole	Create a role with a trust policy as defined in the AWS Data Pipeline Developer Guide	AmazonEC2RoleforDataPipelineRole

Developer power user job function

AWS managed policy name: PowerUserAccess

Use case: This user performs application development tasks and can create and configure resources and services that support AWS aware application development.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: The first statement of this policy uses the `NotAction` element to allow all actions for all AWS services and for all resources except AWS Identity and Access Management, AWS Organizations, and AWS Account Management. The second statement grants IAM permissions to create a service-linked role. This is required by some services that must access resources in another service, such as an Amazon S3 bucket.

It also grants AWS Organizations permissions to view information about the user's organization, including the management account email and organization limitations. Although this policy limits IAM, AWS Organizations, it allows the user to perform all IAM Identity Center actions if IAM Identity Center is enabled. It also grants Account Management permissions to view which AWS Regions are enabled or disabled for the account.

Network administrator job function

AWS managed policy name: NetworkAdministrator

Use case: This user is tasked with setting up and maintaining AWS network resources.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to create and maintain network resources in Auto Scaling, Amazon EC2, AWS Direct Connect, Route 53, Amazon CloudFront, Elastic Load Balancing, AWS Elastic Beanstalk, Amazon SNS, CloudWatch, CloudWatch Logs, Amazon S3, IAM, and Amazon Virtual Private Cloud. For more information about the managed policy, see NetworkAdministrator in *AWS Managed Policy Reference Guide*.

This job function requires the ability to pass roles to AWS services. The policy grants `iam:GetRole` and `iam:PassRole` for only those roles named in the following table. For more information, see Creating roles and attaching policies (console) later in this topic.

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
Allows Amazon VPC to create and manage logs in CloudWatch Logs on the user's behalf to monitor IP traffic going in and out of your VPC	flow-logs-*	Create a role with a trust policy as defined in the Amazon VPC User Guide	This use case does not have an existing AWS managed policy, but the documentation lists the required permissions. See Amazon VPC User Guide.

Read-only access

AWS managed policy name: ReadOnlyAccess

Use case: This user requires read-only access to every resource in an AWS account.

> **Important**
>
> This user will also have access to read data in storage services like Amazon S3 buckets and Amazon DynamoDB tables.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to list, get, describe, and otherwise view resources and their attributes. It does not include mutating functions like create or delete. This policy does include read-only access to security-related AWS services, such as AWS Identity and Access Management and AWS Billing and Cost Management.

View the policy for the full list of services and actions that this policy supports. For more information about the managed policy, see ReadOnlyAccess in *AWS Managed Policy Reference Guide*. If you need a similar policy that does not grant access to read data in storage services, see View-only user job function.

MCP service actions full access

AWS managed policy name: AWSMcpServiceActionsFullAccess

Use case: This user requires access to AWS services using AWS MCP servers. This policy does not grant access to actions taken by an MCP service to other AWS services.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to call any AWS MCP service action. You can use when you do not need to specify permissions per AWS MCP service. It does not grant permissions to actions taken by the MCP service to other AWS services, those permissions must always be granted separately and in addition to MCP service actions. For more information about the managed policy, see AWSMcpServiceActionsFullAccess in *AWS Managed Policy Reference Guide*.

Security auditor job function

AWS managed policy name: SecurityAudit

Use case: This user monitors accounts for compliance with security requirements. This user can access logs and events to investigate potential security breaches or potential malicious activity.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to view configuration data for many AWS services and to review their logs. For more information about the managed policy, see SecurityAudit in *AWS Managed Policy Reference Guide*.

Support user job function

AWS managed policy name: AWSSupportAccess

Use case: This user contacts AWS Support, creates support cases, and views the status of existing cases.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to create and update Support cases. For more information about the managed policy, see AWSSupportAccess in *AWS Managed Policy Reference Guide*.

System administrator job function

AWS managed policy name: SystemAdministrator

Use case: This user sets up and maintains resources for development operations.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants permissions to create and maintain resources across a large variety of AWS services, including AWS CloudTrail, Amazon CloudWatch, AWS CodeCommit, AWS CodeDeploy, AWS Config, AWS Directory Service, Amazon EC2, AWS Identity and Access Management, AWS Key Management Service, AWS Lambda, Amazon RDS, Route 53, Amazon S3, Amazon SES, Amazon SQS, AWS Trusted Advisor, and Amazon VPC. For more information about the managed policy, see SystemAdministrator in *AWS Managed Policy Reference Guide*.

This job function requires the ability to pass roles to AWS services. The policy grants `iam:GetRole` and `iam:PassRole` for only those roles named in the following table. For more information, see Creating roles and attaching policies (console) later in this topic. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
Allow apps running in EC2 instances in an Amazon ECS	ecr-sysadmin-*	Amazon EC2 Role for EC2 Container Service	AmazonEC2 ContainerServiceforEC2Role

Use case	Role name (* is a wildcard)	Service role type to select	AWS managed policy to select
cluster to access Amazon ECS			
Allow a user to monitor databases	rds-monitoring-role	Amazon RDS Role for Enhanced Monitoring	AmazonRDSEnhancedMonitoringRole
Allow apps running in EC2 instances to access AWS resources.	ec2-sysadmin-*	Amazon EC2	Sample policy for role that grants access to an S3 bucket as shown in the Amazon EC2 User Guide; customize as needed
Allow Lambda to read DynamoDB streams and write to CloudWatch Logs	lambda-sysadmin-*	AWS Lambda	AWSLambdaDynamoDBExecutionRole

View-only user job function

AWS managed policy name: ViewOnlyAccess

Use case: This user can view a list of AWS resources and basic metadata in the account across services. The user cannot read resource content or metadata that goes beyond the quota and list information for resources.

Policy updates: AWS maintains and updates this policy. For a history of changes for this policy, view the policy in the IAM console and then choose the **Policy versions** tab. For more information about job function policy updates, see Updates to AWS managed policies for job functions.

Policy description: This policy grants `List*`, `Describe*`, `Get*`, `View*`, and `Lookup*` access to resources for AWS services. To see what actions this policy includes for each service, see ViewOnlyAccess. For more information about the managed policy, see ViewOnlyAccess in *AWS Managed Policy Reference Guide*.

Updates to AWS managed policies for job functions

These policies are all maintained by AWS and are kept up to date to include support for new services and new capabilities as they are added by AWS services. These policies cannot be modified by customers. You can make a copy of the policy and then modify the copy, but that copy is not automatically updated as AWS introduces new services and API operations.

For a job function policy, you can view the version history and the time and date of each update in the IAM console. To do this, use the links on this page to view the policy details. Then choose the **Policy versions** tab to view the versions. This page shows the last 25 versions of a policy. To view all of the versions for a policy, call the get-policy-version AWS CLI command or the GetPolicyVersion API operation.

> **Note**
>
> You can have up to five versions of a customer managed policy, but AWS retains the full version history of AWS managed policies.

Creating roles and attaching policies (console)

Several of the previously listed policies grant the ability to configure AWS services with roles that enable those services to perform operations on your behalf. The job function policies either specify exact role names that you must use or at least include a prefix that specifies the first part of the name that can be used. To create one of these roles, perform the steps in the following procedure.

To create a role for an AWS service (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose a service, and then choose the use case. Use cases are defined by the service to include the trust policy that the service requires.
5. Choose **Next**.
6. For **Permissions policies**, the options depend on the use case that you selected:

   - If the service defines the permissions for the role, you can't select permissions policies.

- Select from a limited set of permission polices.
- Select from all permission policies.
- Select no permissions policies, create the policies after the role is created, and then attach the policies to the role.

7. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

   a. Open the **Set permissions boundary** section, and then choose **Use a permissions boundary to control the maximum role permissions**.

      IAM includes a list of the AWS managed and customer-managed policies in your account.

   b. Select the policy to use for the permissions boundary.

8. Choose **Next**.
9. For **Role name**, the options depend on the service:

   - If the service defines the role name, you can't edit the role name.
   - If the service defines a prefix for the role name, you can enter an optional suffix.
   - If the service doesn't define the role name, you can name the role.

   > **Important**
   >
   > When you name a role, note the following:
   >
   > - Role names must be unique within your AWS account, and can't be made unique by case.
   >
   >   For example, don't create roles named both **`PRODROLE`** and **`prodrole`**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.
   > - You can't edit the name of the role after it's created because other entities might reference the role.

10. (Optional) For **Description**, enter a description for the role.
11. (Optional) To edit the use cases and permissions for the role, in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.

12. (Optional) To help identify, organize, or search for the role, add tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources in the *IAM User Guide*.
13. Review the role, and then choose **Create role**.

Example 1: Configuring a user as a database administrator (console)

This example shows the steps required to configure Alice, an IAM user, as a Database Administrator. You use the information in first row of the table in that section and allow the user to enable Amazon RDS monitoring. You attach the DatabaseAdministrator policy to Alice's IAM user so that they can manage the Amazon database services. That policy also allows Alice to pass a role called `rds-monitoring-role` to the Amazon RDS service that allows the service to monitor the Amazon RDS databases on their behalf.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Policies**, type **`database`** in the search box, and then press enter.
3. Select the radio button for the **DatabaseAdministrator** policy, choose **Actions**, and then choose **Attach**.
4. In the list of entities, select **Alice** and then choose **Attach policy**. Alice now can administer AWS databases. However, to allow Alice to monitor those databases, you must configure the service role.
5. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
6. Choose the **AWS Service** role type, and then choose **Amazon RDS**.
7. Choose the **Amazon RDS Role for Enhanced Monitoring** use case.
8. Amazon RDS defines the permissions for your role. Choose **Next: Review** to continue.
9. The role name must be one of those specified by the DatabaseAdministrator policy that Alice now has. One of those is **`rds-monitoring-role`**. Enter that for the **Role name**.
10. (Optional) For **Role description**, enter a description for the new role.
11. After you review the details, choose **Create role**.
12. Alice can now enable **RDS Enhanced Monitoring** in the **Monitoring** section of the Amazon RDS console. For example, they might do this when they create a DB instance, create a read replica, or modify a DB instance. They must enter the role name they created (rds-monitoring-role) in the **Monitoring Role** box when they set **Enable Enhanced Monitoring** to **Yes**.

Example 2: Configuring a user as a network administrator (console)

This example shows the steps required to configure Jorge, an IAM user, as a Network Administrator. It uses the information in the table in that section to allow Jorge to monitor IP traffic going to and from a VPC. It also allows Jorge to capture that information in the logs in CloudWatch Logs. You attach the NetworkAdministrator policy to Jorge's IAM user so that they can configure AWS network resources. That policy also allows Jorge to pass a role whose name begins with `flow-logs*` to Amazon EC2 when you create a flow log. In this scenario, unlike Example 1, there isn't a predefined service role type, so you must perform a few steps differently.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies** and then enter **`network`** in the search box, and then press enter.
3. Select the radio button next to **NetworkAdministrator** policy, choose **Actions**, and then choose **Attach**.
4. In the list of users, select the checkbox next to **Jorge** and then choose **Attach policy**. Jorge can now administer AWS network resources. However, to enable monitoring of IP traffic in your VPC, you must configure the service role.
5. Because the service role you need to create doesn't have a predefined managed policy, you must first create it. In the navigation pane, choose **Policies**, then choose **Create policy**.
6. In the **Policy editor** section, choose the **JSON** option and copy the text from the following JSON policy document. Paste this text into the **JSON** text box.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
```

```
      "Resource": "*"
    }
  ]
}
```

7. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options any time. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring.

8. On the **Review and create** page, type **`vpc-flow-logs-policy-for-service-role`** for the policy name. Review the **Permissions defined in this policy** to see the permissions granted by your policy, and then choose **Create policy** to save your work.

   The new policy appears in the list of managed policies and is ready to attach.

9. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
10. Choose the **AWS Service** role type, and then choose **Amazon EC2**.
11. Choose the **Amazon EC2** use case.
12. On the **Attach permissions policies** page, choose the policy you created earlier, **vpc-flow-logs-policy-for-service-role**, and then choose **Next: Review**.
13. The role name must be permitted by the NetworkAdministrator policy that Jorge now has. Any name that begins with `flow-logs-` is allowed. For this example, enter **`flow-logs-for-jorge`** for the **Role name**.
14. (Optional) For **Role description**, enter a description for the new role.
15. After you review the details, choose **Create role**.
16. Now you can configure the trust policy required for this scenario. On the **Roles** page, choose the **flow-logs-for-jorge** role (the name, not the checkbox). On the details page for your new role, choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
17. Change the "Service" line to read as follows, replacing the entry for `ec2.amazonaws.com`:

```
      "Service": "vpc-flow-logs.amazonaws.com"
```

18. Jorge can now create flow logs for a VPC or subnet in the Amazon EC2 console. When you create the flow log, specify the **flow-logs-for-jorge** role. That role has the permissions to create the log and write data to it.

AWS global condition context keys

When a principal makes a request to AWS, AWS gathers the request information into a request context. You can use the `Condition` element of a JSON policy to compare keys in the request context with key values that you specify in your policy. Request information is provided by different sources, including the principal making the request, the resource the request is made against, and the metadata about the request itself.

Global condition keys can be used across all AWS services. While these condition keys can be used in all policies, the key is not available in every request context. For example, the `aws:SourceAccount` condition key is only available when the call to your resource is made directly by an AWS service principal. To learn more about the circumstances under which a global key is included in the request context, see the **Availability** information for each key.

Some individual services create their own condition keys that are available in the request context for other services. **Cross-service condition keys** are a type of global condition key that include a prefix matching the name of the service, such as `ec2:` or `lambda:`, but are available across other services.

Service-specific condition keys are defined for use with an individual AWS service. For example, Amazon S3 lets you write a policy with the `s3:VersionId` condition key to limit access to a specific version of an Amazon S3 object. This condition key is unique to the service, meaning it only works with requests to the Amazon S3 service. For condition keys that are service-specific, see Actions, Resources, and Condition Keys for AWS Services and choose the service whose keys you want to view.

> **Note**
>
> If you use condition keys that are available only in some circumstances, you can use the IfExists versions of the condition operators. If the condition keys are missing from a request context, the policy can fail the evaluation. For example, use the following condition block with `...IfExists` operators to match when a request comes from a specific IP range or from a specific VPC. If either or both keys are not included in the request context, the condition still returns `true`. The values are only checked if the specified key is included in

the request context. For more information about how a policy is evaluated when a key is not present for other operators, see Condition operators.

```
"Condition": {
    "IpAddressIfExists": {"aws:SourceIp" : ["xxx"] },
    "StringEqualsIfExists" : {"aws:SourceVpc" : ["yyy"]}
}
```

Important

To compare your condition against a request context with multiple key values, you must use the `ForAllValues` or `ForAnyValue` set operators. Use set operators only with multivalued condition keys. Do not use set operators with single-valued condition keys. For more information, see Set operators for multivalued context keys.

Properties of the principal	Properties of a role session	Properties of the network	Properties of the resource	Properties of the request
aws:PrincipalArn	aws:AssumedRoot	aws:SourceIp	aws:ResourceAccount	aws:CalledVia
aws:PrincipalAccount	aws:FederatedProvider	aws:SourceVpc	aws:ResourceOrgID	aws:CalledViaFirst
aws:PrincipalOrgPaths	aws:CrossPartitionSourceRegion	aws:SourceVpcArn	aws:ResourceOrgPaths	aws:CalledViaLast
aws:PrincipalOrgID	aws:TokenIssueTime	aws:SourceVpce	aws:ResourceTag/ *tag-key*	aws:CalledViaAWSMCP
aws:PrincipalTag/ *tag-key*	aws:SignInSessionArn	aws:VpceAccount		aws:ViaAWSService
		aws:VpceOrgID		aws:ViaAWSMCPService

Properties of the principal	Properties of a role session	Properties of the network	Properties of the resource	Properties of the request
aws:PrincipalIsAWSService aws:PrincipalServiceName aws:PrincipalServiceNamesList aws:PrincipalType aws:userid aws:username	aws:MultiFactorAuthAge aws:MultiFactorAuthPresent aws:ChatbotSourceArn aws:Ec2InstanceSourceVpc aws:Ec2InstanceSourcePrivateIPv4 aws:SourceIdentity aws:ViaCustomerDomain ec2:RoleDelivery ec2:SourceInstanceArn	aws:VpceOrgPaths aws:VpcSourceIp		aws:CurrentTime aws:EpochTime aws:referer aws:RequestedRegion aws:RequestTag/ *tag-key* aws:TagKeys aws:SecureTransport aws:SourceAccount aws:SourceArn aws:SourceOrgID aws:SourceOrgPaths aws:UserAgent aws:IsMcpServiceAction

Properties of the principal	Properties of a role session	Properties of the network	Properties of the resource	Properties of the request
	glue:RoleAssumedBy glue:CredentialIssuingService codebuild:BuildArn codebuild:ProjectArn lambda:SourceFunctionArn ssm:SourceInstanceArn identitystore:UserId			

Sensitive condition keys

The following condition keys are considered sensitive. The use of wildcards in these condition keys does not have any valid use cases, even with a substring of the key value with a wildcard. This is because the wildcard might match the condition key to any value, which could pose a security risk.

- aws:PrincipalAccount
- aws:PrincipalOrgID

- aws:ResourceAccount
- aws:ResourceOrgID
- aws:SourceAccount
- aws:SourceOrgID
- aws:SourceVpc
- aws:SourceVpce
- aws:VpceAccount
- aws:VpceOrgID

Properties of the principal

Use the following condition keys to compare details about the principal making the request with the principal properties that you specify in the policy. For a list of principals that can make requests, see How to specify a principal.

aws:PrincipalArn

Use this key to compare the Amazon Resource Name (ARN) of the principal that made the request with the ARN that you specify in the policy. For IAM roles, the request context returns the ARN of the role, not the ARN of the user that assumed the role.

- **Availability** – This key is included in the request context for all signed requests. Anonymous requests do not include this key. You can specify the following types of principals in this condition key:
  - IAM role
  - IAM user
  - AWS STS federated user principal
  - AWS account root user
- **Data type** – ARN

  AWS recommends that you use ARN operators instead of string operators when comparing ARNs.
- **Value type** – Single-valued
- **Example values** The following list shows the request context value returned for different types of principals that you can specify in the `aws:PrincipalArn` condition key:

- **IAM role** – The request context contains the following value for condition key `aws:PrincipalArn`. Do not specify the assumed role session ARN as a value for this condition key. For more information about the assumed role session principal, see Role session principals.

```
arn:aws:iam::123456789012:role/role-name
```

- **IAM user** – The request context contains the following value for condition key `aws:PrincipalArn`.

```
arn:aws:iam::123456789012:user/user-name
```

- **AWS STS federated user principals** – The request context contains the following value for condition key `aws:PrincipalArn`.

```
arn:aws:sts::123456789012:federated-user/user-name
```

- **AWS account root user** – The request context contains the following value for condition key `aws:PrincipalArn`. When you specify the root user ARN as the value for the `aws:PrincipalArn` condition key, it limits permissions only for the root user of the AWS account. This is different from specifying the root user ARN in the principal element of a resource-based policy, which delegates authority to the AWS account. For more information about specifying the root user ARN in the principal element of a resource-based policy, see AWS account principals.

```
arn:aws:iam::123456789012:root
```

You can specify the root user ARN as a value for condition key `aws:PrincipalArn` in AWS Organizations service control policies (SCPs). SCPs are a type of organization policy used to manage permissions in your organization and affect only member accounts in the organization. An SCP restricts permissions for IAM users and roles in member accounts, including the member account's root user. For more information about the effect of SCPs on permissions, see SCP effects on permissions in the *AWS Organizations User Guide*.

aws:PrincipalAccount

Use this key to compare the account to which the requesting principal belongs with the account identifier that you specify in the policy. For anonymous requests, the request context returns `anonymous`.

- **Availability** – This key is included in the request context for all requests, including anonymous requests.
- **Data type** – String
- **Value type** – Single-valued

In the following example, access is denied except to principals with the account number `123456789012`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessFromPrincipalNotInSpecificAccount",
      "Action": "service:*",
      "Effect": "Deny",
      "Resource": [
        "arn:aws:service:us-east-1:111122223333:resource"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalAccount": [
            "123456789012"
          ]
        }
      }
    }
  ]
}
```

aws:PrincipalOrgPaths

Use this key to compare the AWS Organizations path for the principal who is making the request to the path in the policy. That principal can be an IAM user, IAM role, AWS STS federated user principal, or AWS account root user. In a policy, this condition key ensures that the requester is an account member within the specified organization root or organizational units (OUs) in AWS Organizations. An AWS Organizations path is a text representation of the structure of an AWS Organizations entity. For more information about using and understanding paths, see Understand the AWS Organizations entity path.

- **Availability** – This key is included in the request context only if the principal is a member of an organization. Anonymous requests do not include this key.
- **Data type** – String (list)
- **Value type** – Multivalued

> **Note**
>
> Organization IDs are globally unique but OU IDs and root IDs are unique only within an organization. This means that no two organizations share the same organization ID. However, another organization might have an OU or root with the same ID as yours. We recommend that you always include the organization ID when you specify an OU or root.

For example, the following condition returns `true` for principals in accounts that are attached directly to the `ou-ab12-22222222` OU, but not in its child OUs.

```
"Condition" : { "ForAnyValue:StringEquals" : {
     "aws:PrincipalOrgPaths":["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/"]
}}
```

The following condition returns `true` for principals in an account that is attached directly to the OU or any of its child OUs. When you include a wildcard, you must use the `StringLike` condition operator.

```
"Condition" : { "ForAnyValue:StringLike" : {
     "aws:PrincipalOrgPaths":["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/
*"]
```

```
}}
```

The following condition returns `true` for principals in an account that is attached directly to any of the child OUs, but not directly to the parent OU. The previous condition is for the OU or any children. The following condition is for only the children (and any children of those children).

```
"Condition" : { "ForAnyValue:StringLike" : {
     "aws:PrincipalOrgPaths":["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/
ou-*"]
}}
```

The following condition allows access for every principal in the `o-a1b2c3d4e5` organization, regardless of their parent OU.

```
"Condition" : { "ForAnyValue:StringLike" : {
     "aws:PrincipalOrgPaths":["o-a1b2c3d4e5/*"]
}}
```

`aws:PrincipalOrgPaths` is a multivalued condition key. Multivalued keys can have multiple values in the request context. When you use multiple values with the `ForAnyValue` condition operator, the principal's path must match one of the paths listed in the policy. For more information about multivalued condition keys, see Set operators for multivalued context keys.

```
    "Condition": {
        "ForAnyValue:StringLike": {
            "aws:PrincipalOrgPaths": [
                "o-a1b2c3d4e5/r-ab12/ou-ab12-33333333/*",
                "o-a1b2c3d4e5/r-ab12/ou-ab12-22222222/*"
            ]
        }
    }
```

aws:PrincipalOrgID

Use this key to compare the identifier of the organization in AWS Organizations to which the requesting principal belongs with the identifier specified in the policy.

- **Availability** – This key is included in the request context only if the principal is a member of an organization. Anonymous requests do not include this key.
- **Data type** – String

- **Value type** – Single-valued

This global key provides an alternative to listing all the account IDs for all AWS accounts in an organization. You can use this condition key to simplify specifying the `Principal` element in a resource-based policy. You can specify the organization ID in the condition element. When you add and remove accounts, policies that include the `aws:PrincipalOrgID` key automatically include the correct accounts and don't require manual updating.

For example, the following Amazon S3 bucket policy allows members of any account in the `o-xxxxxxxxxxx` organization to add an object into the `amzn-s3-demo-bucket` bucket.

JSON

```
{
 "Version":"2012-10-17",
 "Statement": {
   "Sid": "AllowPutObject",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "s3:PutObject",
   "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
   "Condition": {"StringEquals":
     {"aws:PrincipalOrgID":"o-xxxxxxxxxxx"}
   }
 }
}
```

Note

This global condition also applies to the management account of an AWS organization. This policy prevents all principals outside of the specified organization from accessing the Amazon S3 bucket. This includes any AWS services that interact with your internal resources, such as AWS CloudTrail sending log data to your Amazon S3 buckets. To learn how you can safely grant access for AWS services, see aws:PrincipalIsAWSService.

For more information about AWS Organizations, see What Is AWS Organizations? in the *AWS Organizations User Guide*.

aws:PrincipalTag/*tag-key*

Use this key to compare the tag attached to the principal making the request with the tag that you specify in the policy. If the principal has more than one tag attached, the request context includes one `aws:PrincipalTag` key for each attached tag key.

- **Availability** – This key is included in the request context if the principal is using an IAM user with attached tags. It is included for a principal using an IAM role with attached tags or session tags. Anonymous requests do not include this key.
- **Data type** – String
- **Value type** – Single-valued

You can add custom attributes to a user or role in the form of a key-value pair. For more information about IAM tags, see Tags for AWS Identity and Access Management resources. You can use `aws:PrincipalTag` to control access for AWS principals.

This example shows how you might create an identity-based policy that allows users with the **`department=hr`** tag to manage IAM users, groups, or roles. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:Get*",
        "iam:List*",
        "iam:Generate*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/department": "hr"
        }
      }
    }
```

```
  ]
}
```

aws:PrincipalIsAWSService

Use this key to check whether the call to your resource is being made directly by an AWS service principal. For example, AWS CloudTrail uses the service principal `cloudtrail.amazonaws.com` to write logs to your Amazon S3 bucket. The request context key is set to true when a service uses a service principal to perform a direct action on your resources. The context key is set to false if the service uses the credentials of an IAM principal to make a request on the principal's behalf. It is also set to false if the service uses a service role or service-linked role to make a call on the principal's behalf.

- **Availability** – This key is present in the request context for all signed API requests that use AWS credentials. Anonymous requests do not include this key.
- **Data type** – Boolean
- **Value type** – Single-valued

You can use this condition key to limit access to your trusted identities and expected network locations while safely granting access to AWS services.

In the following Amazon S3 bucket policy example, access to the bucket is restricted unless the request originates from `vpc-111bbb22` or is from a service principal, such as CloudTrail.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ExpectedNetworkServicePrincipal",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1/AWSLogs/AccountNumber/*",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": "vpc-111bbb22"
```

```
        },
        "BoolIfExists": {
          "aws:PrincipalIsAWSService": "false"
        }
      }
    }
  ]
}
```

In the following video, learn more about how you might use the `aws:PrincipalIsAWSService` condition key in a policy.

Safely grant access to your authorized users, expected network locations, and AWS services together.

aws:PrincipalServiceName

Use this key to compare the service principal name in the policy with the service principal that is making requests to your resources. You can use this key to check whether this call is made by a specific service principal. When a service principal makes a direct request to your resource, the `aws:PrincipalServiceName` key contains the name of the service principal. For example, the AWS CloudTrail service principal name is `cloudtrail.amazonaws.com`.

- **Availability** – This key is present in the request when the call is made by an AWS service principal. This key is not present in any other situation, including the following:
  - If the service uses a service role or service-linked role to make a call on the principal's behalf.
  - If the service uses the credentials of an IAM principal to make a request on the principal's behalf.
  - If the call is made directly by an IAM principal.
  - If the call is made by an anonymous requester.
- **Data type** – String
- **Value type** – Single-valued

You can use this condition key to limit access to your trusted identities and expected network locations while safely granting access to an AWS service.

In the following Amazon S3 bucket policy example, access to the bucket is restricted unless the request originates from `vpc-111bbb22` or is from a service principal, such as CloudTrail.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ExpectedNetworkServicePrincipal",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1/AWSLogs/AccountNumber/*",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": "vpc-111bbb22",
          "aws:PrincipalServiceName": "cloudtrail.amazonaws.com"
        }
      }
    }
  ]
}
```

aws:PrincipalServiceNamesList

This key provides a list of all service principal names that belong to the service. This is an advanced condition key. You can use it to restrict the service from accessing your resource from a specific Region only. Some services might create Regional service principals to indicate a particular instance of the service within a specific Region. You can limit access to a resource to a particular instance of the service. When a service principal makes a direct request to your resource, the `aws:PrincipalServiceNamesList` contains an unordered list of all service principal names associated with the Regional instance of the service.

- **Availability** – This key is present in the request when the call is made by an AWS service principal. This key is not present in any other situation, including the following:
  - If the service uses a service role or service-linked role to make a call on the principal's behalf.
  - If the service uses the credentials of an IAM principal to make a request on the principal's behalf.
  - If the call is made directly by an IAM principal.

  - If the call is made by an anonymous requester.
- **Data type** – String (list)
- **Value type** – Multivalued

`aws:PrincipalServiceNamesList` is a multivalued condition key. Multivalued keys can have multiple values in the request context. You must use the `ForAnyValue` or `ForAllValues` set operators with string condition operators for this key. For more information about multivalued condition keys, see Set operators for multivalued context keys.

aws:PrincipalType

Use this key to compare the type of principal making the request with the principal type that you specify in the policy. For more information, see How to specify a principal. For specific examples of `principal` key values, see Principal key values.

- **Availability** – This key is included in the request context for all requests, including anonymous requests.
- **Data type** – String
- **Value type** – Single-valued

aws:userid

Use this key to compare the requester's principal identifier with the ID that you specify in the policy. For IAM users, the request context value is the user ID. For IAM roles, this value format can vary. For details about how the information appears for different principals, see How to specify a principal. For specific examples of `principal` key values, see Principal key values.

- **Availability** – This key is included in the request context for all requests, including anonymous requests.
- **Data type** – String
- **Value type** – Single-valued

aws:username

Use this key to compare the requester's user name with the user name that you specify in the policy. For details about how the information appears for different principals, see How to specify a principal. For specific examples of `principal` key values, see Principal key values.

- **Availability** – This key is always included in the request context for IAM users. Anonymous requests and requests that are made using the AWS account root user or IAM roles do not include this key. Requests made using IAM Identity Center credentials do not include this key in the context.
- **Data type** – String
- **Value type** – Single-valued

Properties of a role session

Use the following condition keys to compare properties of the role session at the time the session was generated. These condition keys are only available when a request is made by a principal with role session or federated user principal credentials. The values for these condition keys are embedded in the role's session token.

A role is a type of principal. You can also use the condition keys from the Properties of the principal section to evaluate the properties of a role when a role is making a request.

aws:AssumedRoot

Use this key to check whether the request was made using AssumeRoot. `AssumeRoot` returns short term credentials for a privileged root user session you can use to take privileged actions on member accounts in your organization. For more information, see Centrally manage root access for member accounts.

- **Availability** – This key is included in the request context only when the principal uses credentials from AssumeRoot to make the request.
- **Data type** – Boolean
- **Value type** – Single-valued

In the following example, when used as a service control policy, denies the usage of the long term credentials of a root user in an AWS Organizations member account. The policy does not deny `AssumeRoot` sessions from taking the actions allowed by an `AssumeRoot` session.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement":[
       {
          "Effect":"Deny",
          "Action":"*",
          "Resource": "*",
          "Condition":{
             "ArnLike":{
                "aws:PrincipalArn":[
                   "arn:aws:iam::*:root"
                ]
             },
             "Null":{
                "aws:AssumedRoot":"true"
             }
          }
       }
    ]
}
```

aws:FederatedProvider

Use this key to compare the principal's issuing identity provider (IdP) with the IdP that you specify in the policy. This key applies to an IAM role assumed using the AssumeRoleWithWebIdentity AWS STS operation. When the resulting role session's temporary credentials are used to make a request, the request context identifies the IdP that authenticated the original federated identity.

- **Availability** – This key is present in the role-session of a role that was assumed using OpenID Connect (OIDC) provider, and in the role-trust policy when an OIDC provider is used to call `AssumeRoleWithWebIdentity`.
- **Data type** – String*
- **Value type** – Single-valued

* The data type depends on your IdP:

- If you're using a built-in AWS IdP, like Amazon Cognito, the key value will be a **string**. The key value might look like: `cognito-identity.amazonaws.com`.
- If you're using an IdP that is not built-in to AWS, like GitHub or Amazon EKS, the key value will be **ARN**. The key value might look like: `arn:aws:iam::`*`111122223333`*`:oidc-provider/oidc.eks.`*`region`*`.amazonaws.com/id/`*`OIDC_Provider_ID`*.

- If you're using cross-partition identity federation, the key value is the **ARN** of the OIDC identity provider in the destination account that trusts the source account's issuer URL. The key value might look like: `arn:aws-cn:iam::111122223333:oidc-provider/issuer-id.tokens.sts.global.api.aws`.

For more information on external IdPs and `AssumeRoleWithWebIdentity`, see Common scenarios. For more information, see Role session principals.

aws:CrossPartitionSourceRegion

Use this key to compare the AWS Region where AWS STS issued a web identity token with the AWS Region that you specify in the policy. When you assume a role across an AWS partition boundary using a token from the GetWebIdentityToken operation, this key holds the source AWS Region where the token was issued. After a policy change propagates, an explicit deny using this key blocks matching requests from existing sessions without waiting for their temporary credentials to expire.

- **Availability** – This key is present in the request context only for role sessions created through cross-partition identity federation by calling AssumeRoleWithWebIdentity with a token issued by `GetWebIdentityToken`. It is not present for sessions created with tokens from other OpenID Connect (OIDC) identity providers.
- **Data type** – String
- **Value type** – Single-valued
- **Policy types** – Use this key in policies that are evaluated for requests made with the role session, including SCPs, RCPs for supported resources, identity-based policies, and resource-based policies. It is not available when the role trust policy is evaluated for the `AssumeRoleWithWebIdentity` call. To restrict the source AWS Region before credentials are returned, use the *issuer-host*`:source_region` condition key in the role trust policy.
- **Role chaining** – This key is not transitive. It is not present in sessions created when the role session assumes another role.

The following example SCP denies every request made with credentials whose token was issued in `us-east-1`.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "DenySourceRegion",
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:CrossPartitionSourceRegion": "us-east-1"
                }
            }
        }
    ]
}
```

aws:TokenIssueTime

Use this key to compare the date and time that temporary security credentials were issued with the date and time that you specify in the policy.

- **Availability** – This key is included in the request context only when the principal uses temporary credentials to make the request. The key is not present in AWS CLI, AWS API, or AWS SDK requests that are made using access keys.
- **Data type** – Date
- **Value type** – Single-valued

To learn which services support using temporary credentials, see AWS services that work with IAM.

aws:SignInSessionArn

Use this key to compare your AWS Sign-In OAuth session with the session that you specify in the policy.

When you use an OAuth-based flow such as AWS CLI login (`aws login`) or AWS MCP Server, AWS Sign-In includes a sign-in session ARN in the issued credentials. AWS Sign-In propagates that ARN to subsequent requests made using OAuth access tokens. With this key, you can correlate API activity with the originating sign-in session and apply IAM policies to individual sessions.

For interactive OAuth flows, the refresh token determines the sign-in session ARN lifecycle. A new access token issued during a refresh continues to carry the same sign-in session ARN.

For non-interactive OAuth flows, the access token determines the sign-in session ARN lifecycle. Each new access token contains a new sign-in session ARN.

- **Availability** – This key is included in the request context when the request originates from an AWS Sign-In OAuth session. This key is not available for console sessions.
- **Data type** – ARN
- **Value type** – Single-valued

Use ARN operators instead of string operators when you compare ARNs.

The following policy denies all actions associated with a specific AWS Sign-In session, although requests from other active sessions continue:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenySpecificSignInSession",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "aws:SignInSessionArn": "arn:aws:signin:us-east-1:111122223333:session/
session-id"
        }
      }
    }
  ]
}
```

aws:MultiFactorAuthAge

Use this key to compare the number of seconds since the requesting principal was authorized using MFA with the number that you specify in the policy. For more information about MFA, see AWS Multi-factor authentication in IAM.

> **Important**
>
> This condition key is not present for federated identities or requests made using access keys to sign AWS CLI, AWS API, or AWS SDK requests. To learn more about adding MFA protection to API operations with temporary security credentials, see Secure API access with MFA.
> To check whether MFA is used to validate IAM federated identities, you can pass the authentication method from your identity provider to AWS as a session tag. For details, see Pass session tags in AWS STS. To enforce MFA for IAM Identity Center identities, you can enable attributes for access control to pass a SAML assertion claim with the authentication method from your identity provider to IAM Identity Center.

- **Availability** – This key is included in the request context only when the principal uses temporary security credentials to make the request. Policies with MFA conditions can be attached to:
  - An IAM user or group
  - A resource such as an Amazon S3 bucket, Amazon SQS queue, or Amazon SNS topic
  - The trust policy of an IAM role that can be assumed by a user
- **Data type** – Numeric
- **Value type** – Single-valued

aws:MultiFactorAuthPresent

Use this key to check whether multi-factor authentication (MFA) was used to validate the temporary security credentials that made the request.

> **Important**
>
> This condition key is not present for federated identities or requests made using access keys to sign AWS CLI, AWS API, or AWS SDK requests. To learn more about adding MFA protection to API operations with temporary security credentials, see Secure API access with MFA.
> To check whether MFA is used to validate IAM federated identities, you can pass the authentication method from your identity provider to AWS as a session tag. For details, see Pass session tags in AWS STS. To enforce MFA for IAM Identity Center identities, you can

enable attributes for access control to pass a SAML assertion claim with the authentication method from your identity provider to IAM Identity Center.

- **Availability** – This key is included in the request context only when the principal uses temporary credentials to make the request. Policies with MFA conditions can be attached to:
  - An IAM user or group
  - A resource such as an Amazon S3 bucket, Amazon SQS queue, or Amazon SNS topic
  - The trust policy of an IAM role that can be assumed by a user
- **Data type** – Boolean
- **Value type** – Single-valued

Temporary credentials are used to authenticate IAM roles and IAM users with temporary tokens from AssumeRole or GetSessionToken, and users of the AWS Management Console.

IAM user access keys are long-term credentials, but in some cases, AWS creates temporary credentials on behalf of IAM users to perform operations. In these cases, the `aws:MultiFactorAuthPresent` key is present in the request and set to a value of `false`. There are two common cases where this can happen:

- IAM users in the AWS Management Console unknowingly use temporary credentials. Users sign into the console using their user name and password, which are long-term credentials. However, in the background, the console generates temporary credentials on behalf of the user.
- If an IAM user makes a call to an AWS service, the service re-uses the user's credentials to make another request to a different service. For example, when calling Athena to access an Amazon S3 bucket, or when using CloudFormation to create an Amazon EC2 instance. For the subsequent request, AWS uses temporary credentials.

To learn which services support using temporary credentials, see AWS services that work with IAM.

The `aws:MultiFactorAuthPresent` key is not present when an API or CLI command is called with long-term credentials, such as user access key pairs. Therefore we recommend that when you check for this key that you use the `...IfExists` versions of the condition operators.

It is important to understand that the following `Condition` element is ***not*** a reliable way to check whether a request is authenticated using MFA.

```
#####   WARNING: NOT RECOMMENDED   #####
"Effect" : "Deny",
"Condition" : { "Bool" : { "aws:MultiFactorAuthPresent" : "false" } }
```

This combination of the `Deny` effect, `Bool` element, and `false` value denies requests that can be authenticated using MFA, but were not. This applies only to temporary credentials that support using MFA. This statement does not deny access to requests that are made using long-term credentials, or to requests that are authenticated using MFA. Use this example with caution because its logic is complicated and it does not test whether MFA-authentication was actually used.

Also do not use the combination of the `Deny` effect, `Null` element, and `true` because it behaves the same way and the logic is even more complicated.

Recommended Combination

We recommend that you use the `BoolIfExists` operator to check whether a request is authenticated using MFA.

```
"Effect" : "Deny",
"Condition" : { "BoolIfExists" : { "aws:MultiFactorAuthPresent" : "false" } }
```

This combination of `Deny`, `BoolIfExists`, and `false` denies requests that are not authenticated using MFA. Specifically, it denies requests from temporary credentials that do not include MFA. It also denies requests that are made using long-term credentials, such as AWS CLI or AWS API operations made using access keys. The `*IfExists` operator checks for the presence of the `aws:MultiFactorAuthPresent` key and whether or not it could be present, as indicated by its existence. Use this when you want to deny any request that is not authenticated using MFA. This is more secure, but can break any code or scripts that use access keys to access the AWS CLI or AWS API.

Alternative Combinations

You can also use the `BoolIfExists` operator to allow MFA-authenticated requests and AWS CLI or AWS API requests that are made using long-term credentials.

```
"Effect" : "Allow",
"Condition" : { "BoolIfExists" : { "aws:MultiFactorAuthPresent" : "true" } }
```

This condition matches either if the key exists and is present **or** if the key does not exist. This combination of `Allow`, `BoolIfExists`, and `true` allows requests that are authenticated using

MFA, or requests that cannot be authenticated using MFA. This means that AWS CLI, AWS API, and AWS SDK operations are allowed when the requester uses their long-term access keys. This combination does not allow requests from temporary credentials that could, but do not include MFA.

When you create a policy using the IAM console visual editor and choose **MFA required**, this combination is applied. This setting requires MFA for console access, but allows programmatic access with no MFA.

Alternatively, you can use the `Bool` operator to allow programmatic and console requests only when authenticated using MFA.

```
"Effect" : "Allow",
"Condition" : { "Bool" : { "aws:MultiFactorAuthPresent" : "true" } }
```

This combination of the `Allow`, `Bool`, and `true` allows only MFA-authenticated requests. This applies only to temporary credentials that support using MFA. This statement does not allow access to requests that were made using long-term access keys, or to requests made using temporary credentials without MFA.

Do not use a policy construct similar to the following to check whether the MFA key is present:

```
#####   WARNING: USE WITH CAUTION   #####

"Effect" : "Allow",
"Condition" : { "Null" : { "aws:MultiFactorAuthPresent" : "false" } }
```

This combination of the `Allow` effect, `Null` element, and `false` value allows only requests that can be authenticated using MFA, regardless of whether the request is actually authenticated. This allows all requests that are made using temporary credentials, and denies access for long-term credentials. Use this example with caution because it does not test whether MFA-authentication was actually used.

aws:ChatbotSourceArn

Use this key to compare the source chat configuration ARN set by the principal to the chat configuration ARN you specify in the policy of the IAM role associated with your channel configuration. You can authorize requests based on the assume role session initiated by Amazon Q Developer in chat applications.

- **Availability** – This key is included in the request context by the Amazon Q Developer in chat applications service whenever a role session is assumed. The key value is the chat configuration ARN, such as when you run an AWS CLI command from a chat channel.
- **Data type** – ARN
- **Value type** – Single-valued
- **Example value** – `arn:aws::chatbot::123456789021:chat-configuration/slack-channel/private_channel`

The following policy denies Amazon S3 put requests on the specified bucket for all requests originating from a Slack channel.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleS3Deny",
            "Effect": "Deny",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "ArnLike": {
                      "aws:ChatbotSourceArn": "arn:aws:chatbot::*:chat-
configuration/slack-channel/*"
                }
            }
        }
    ]
}
```

aws:Ec2InstanceSourceVpc

This key identifies the VPC to which Amazon EC2 IAM role credentials were delivered to. You can use this key in a policy with the `aws:SourceVPC` global key to check if a call is made from a VPC (`aws:SourceVPC`) that matches the VPC where a credential was delivered to (`aws:Ec2InstanceSourceVpc`).

- **Availability** – This key is included in the request context whenever the requester is signing requests with an Amazon EC2 role credential. It can be used in IAM policies, service control policies, VPC endpoint policies, and resource policies.
- **Data type** – String
- **Value type** – Single-valued

This key can be used with VPC identifier values, but is most useful when used as a variable combined with the `aws:SourceVpc` context key. The `aws:SourceVpc` context key is included in the request context only if the requester uses a VPC endpoint to make the request. Using `aws:Ec2InstanceSourceVpc` with `aws:SourceVpc` allows you to use `aws:Ec2InstanceSourceVpc` more broadly since it compares values that typically change together.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "RequireSameVPC",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
            "aws:SourceVpc": "${aws:Ec2InstanceSourceVpc}"
        },
        "Null": {
          "ec2:SourceInstanceARN": "false"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

In the example above, access is denied if the `aws:SourceVpc` value isn't equal to the `aws:Ec2InstanceSourceVpc` value. The policy statement is limited to only roles used as Amazon EC2 instance roles by testing for the existence of the `ec2:SourceInstanceARN` condition key.

The policy uses `aws:ViaAWSService` to allow AWS to authorize requests when requests are made on behalf of your Amazon EC2 instance roles. For example, when you make a request from an Amazon EC2 instance to an encrypted Amazon S3 bucket, Amazon S3 makes a call to AWS KMS on your behalf. Some of the keys are not present when the request is made to AWS KMS.

aws:Ec2InstanceSourcePrivateIPv4

This key identifies the private IPv4 address of the primary elastic network interface to which Amazon EC2 IAM role credentials were delivered. You must use this condition key with its companion key `aws:Ec2InstanceSourceVpc` to make sure that you have a globally unique combination of VPC ID and source private IP. Use this key with `aws:Ec2InstanceSourceVpc` to make sure that a request was made from the same private IP address that the credentials were delivered to.

- **Availability** – This key is included in the request context whenever the requester is signing requests with an Amazon EC2 role credential. It can be used in IAM policies, service control policies, VPC endpoint policies, and resource policies.
- **Data type** – IP address
- **Value type** – Single-valued

> **Important**
>
> This key should not be used alone in an `Allow` statement. Private IP addresses are by definition not globally unique. You should use the `aws:Ec2InstanceSourceVpc` key every time you use the `aws:Ec2InstanceSourcePrivateIPv4` key to specify the VPC your Amazon EC2 instance credentials can be used from.

The following example is a service control policy (SCP) that denies access to all resources unless the request arrives through a VPC Endpoint in the same VPC as the as the role credentials. In this example, `aws:Ec2InstanceSourcePrivateIPv4` limits the credential source to a particular instance based on the source IP.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action":  "*",
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:Ec2InstanceSourceVpc": "${aws:SourceVpc}"
                },
                "Null": {
                    "ec2:SourceInstanceARN": "false"
                },
                "BoolIfExists": {
                    "aws:ViaAWSService": "false"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action":  "*",
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:Ec2InstanceSourcePrivateIPv4": "${aws:VpcSourceIp}"
                },
                "Null": {
                    "ec2:SourceInstanceARN": "false"
                },
                "BoolIfExists": {
                    "aws:ViaAWSService": "false"
                }
            }
        }
    ]
}
```

aws:SourceIdentity

Use this key to compare the source identity that was set by the principal with the source identity that you specify in the policy.

- **Availability** – This key is included in the request context after a source identity has been set when a role is assumed using any AWS STS assume-role CLI command, or AWS STS `AssumeRole` API operation.
- **Data type** – String
- **Value type** – Single-valued

You can use this key in a policy to allow actions in AWS by principals that have set a source identity when assuming a role. Activity for the role's specified source identity appears in AWS CloudTrail. This makes it easier for administrators to determine who or what performed actions with a role in AWS.

Unlike `sts:RoleSessionName`, after the source identity is set, the value cannot be changed. It is present in the request context for all actions taken by the role. The value persists into subsequent role sessions when you use the session credentials to assume another role. Assuming one role from another is called role chaining.

The `sts:SourceIdentity` key is present in the request when the principal initially sets a source identity while assuming a role using any AWS STS assume-role CLI command, or AWS STS `AssumeRole` API operation. The `aws:SourceIdentity` key is present in the request for any actions that are taken with a role session that has a source identity set.

The following role trust policy for `CriticalRole` in account `111122223333` contains a condition for `aws:SourceIdentity` that prevents a principal without a source identity that is set to Saanvi or Diego from assuming the role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeRoleIfSourceIdentity",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:role/CriticalRole"},
```

```
            "Action": [
                "sts:AssumeRole",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringLike": {
                    "aws:SourceIdentity": ["Saanvi","Diego"]
                }
            }
        }
    ]
}
```

To learn more about using source identity information, see Monitor and control actions taken with assumed roles.

ec2:RoleDelivery

Use this key to compare the version of the instance metadata service in the signed request with the IAM role credentials for Amazon EC2. The instance metadata service distinguishes between IMDSv1 and IMDSv2 requests based on whether, for any given request, either the PUT or GET headers, which are unique to IMDSv2, are present in that request.

- **Availability** – This key is included in the request context whenever the role session is created by an Amazon EC2 instance.
- **Data type** – Numeric
- **Value type** – Single-valued
- **Example values** – 1.0, 2.0

You can configure the Instance Metadata Service (IMDS) on each instance so that local code or users must use IMDSv2. When you specify that IMDSv2 must be used, IMDSv1 no longer works.

- Instance Metadata Service Version 1 (IMDSv1) – A request/response method
- Instance Metadata Service Version 2 (IMDSv2) – a session-oriented method

For information about how to configure your instance to use IMDSv2, see Configure the instance metadata options.

In the following example, access is denied if the ec2:RoleDelivery value in the request context is 1.0 (IMDSv1). This policy statement can be applied generally because, if the request is not signed by Amazon EC2 role credentials, it has no effect.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
                {
            "Sid": "RequireAllEc2RolesToUseV2",
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "NumericLessThan": {
                    "ec2:RoleDelivery": "2.0"
                }
            }
        }
    ]
}
```

For more information, see Example policies for working with instance metadata.

ec2:SourceInstanceArn

Use this key to compare the ARN of the instance from which the role's session was generated.

- **Availability** – This key is included in the request context whenever the role session is created by an Amazon EC2 instance.
- **Data type** – ARN
- **Value type** – Single-valued
- **Example value** – arn:aws:ec2:us-west-2:111111111111:instance/instance-id

For policy examples, see Allow a specific instance to view resources in other AWS services.

glue:RoleAssumedBy

The AWS Glue service sets this condition key for each AWS API request where AWS Glue makes a request using a service role on the customer's behalf (not by a job or developer endpoint, but directly by the AWS Glue service). Use this key to verify whether a call to an AWS resource came from the AWS Glue service.

- **Availability** – This key is included in the request context when AWS Glue makes a request using a service role on the customer's behalf.
- **Data type** – String
- **Value type** – Single-valued
- **Example value** – This key is always set to `glue.amazonaws.com`.

The following example adds a condition to allow the AWS Glue service to get an object from an Amazon S3 bucket.

```
{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
        "StringEquals": {
            "glue:RoleAssumedBy": "glue.amazonaws.com"
        }
    }
}
```

glue:CredentialIssuingService

The AWS Glue service sets this key for each AWS API request using a service role that comes from a job or developer endpoint. Use this key to verify whether a call to an AWS resource came from an AWS Glue job or developer endpoint.

- **Availability** – This key is included in the request context when AWS Glue makes a request that comes from a job or developer endpoint.
- **Data type** – String
- **Value type** – Single-valued
- **Example value** – This key is always set to `glue.amazonaws.com`.

The following example adds a condition that is attached to an IAM role that is used by an AWS Glue job. This ensures certain actions are allowed/denied based on whether the role session is used for an AWS Glue job runtime environment.

```
{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
        "StringEquals": {
            "glue:CredentialIssuingService": "glue.amazonaws.com"
        }
    }
}
```

codebuild:BuildArn

This key identifies the CodeBuild build ARN that IAM role credentials were delivered to. Use this key to verify whether a call to an AWS resource came from a specific CodeBuild build.

> **Note**
>
> The full value of `codebuild:BuildArn` is not known in advance because it contains the dynamically generated build ID.

- **Availability** – This key is included in the request context whenever a request is made by a role assumed by CodeBuild.
- **Data type** – ARN
- **Value type** – Single-valued
- **Example value** – arn:aws:codebuild:us-east-1:123456789012:build/MyBuildProject:12345678-1234-1234-1234-123456789012

The following example allows a specific CodeBuild build to have `s3:GetObject` access to the specified bucket.

```
{
    "Effect": "Allow",
    "Action": "s3:GetObject",
```

```
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
        "ArnLike": {
            "codebuild:BuildArn": "arn:aws:codebuild:us-east-1:123456789012:build/
MyBuildProject:*"
        }
    }
}
```

codebuild:ProjectArn

This key identifies the CodeBuild project ARN that IAM role credentials were delivered to an CodeBuild build. Use this key to verify whether a call to an AWS resource came from a specific CodeBuild project.

- **Availability** – This key is included in the request context whenever a request is made by a role assumed by CodeBuild.
- **Data type** – ARN
- **Value type** – Single-valued
- **Example value** – arn:aws:codebuild:us-east-1:123456789012:project/MyBuildProject

The following example allows any build from a specific CodeBuild project to have `s3:GetObject` access to the specified bucket.

```
{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
        "ArnEquals": {
            "codebuild:ProjectArn": "arn:aws:codebuild:us-east-1:123456789012:project/
MyBuildProject"
        }
    }
}
```

lambda:SourceFunctionArn

Use this key to identify the Lambda function ARN that IAM role credentials were delivered to. The Lambda service sets this key for each AWS API request that comes from your function's execution

environment. Use this key to verify whether a call to an AWS resource came from a specific Lambda function's code. Lambda also sets this key for some requests that come from outside the execution environment, such as writing logs to CloudWatch and sending traces to X-Ray.

- **Availability** – This key is included in the request context whenever Lambda function code is invoked.
- **Data type** – ARN
- **Value type** – Single-valued
- **Example value** – arn:aws:lambda:us-east-1:123456789012:function:TestFunction

The following example allows one specific Lambda function to have `s3:PutObject` access the specified bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleSourceFunctionArn",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "ArnEquals": {
                    "lambda:SourceFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:source_lambda"
                }
            }
        }
    ]
}
```

For more information, see Working with Lambda execution environment credentials in the *AWS Lambda Developer Guide*.

ssm:SourceInstanceArn

Use this key to identify the AWS Systems Manager managed instance ARN that IAM role credentials were delivered to. This condition key is not present when the request comes from a managed instance with an IAM role associated with an Amazon EC2 instance profile.

- **Availability** – This key is included in the request context whenever role credentials are delivered to an AWS Systems Manager managed instance.
- **Data type** – ARN
- **Value type** – Single-valued
- **Example value** – arn:aws:ec2:us-west-2:111111111111:instance/instance-id

identitystore:UserId

Use this key to compare IAM Identity Center workforce identity in the signed request with the identity specified in the policy.

- **Availability** – This key is included when the caller of the request is a user in IAM Identity Center.
- **Data type** – String
- **Value type** – Single-valued
- **Example value** – 94482488-3041-7026-18f3-be45837cd0e4

You can find the UserId of a user in IAM Identity Center by making a request to the GetUserId API using the AWS CLI, AWS API, or AWS SDK.

Properties of the network

Use the following condition keys to compare details about the network that the request originated from or passed through with the network properties that you specify in the policy.

aws:SourceIp

Use this key to compare the requester's IP address with the IP address that you specify in the policy. The `aws:SourceIp` condition key can only be used for public IP address ranges.

- **Availability** – This key is included in the request context, except when the requester uses a VPC endpoint to make the request.
- **Data type** – IP address

- **Value type** – Single-valued

The `aws:SourceIp` condition key can be used in a policy to allow principals to make requests only from within a specified IP range.

> **Note**
>
> `aws:SourceIp` supports both IPv4 and IPv6 address or range of IP addresses. For a list of AWS services that support IPv6, see AWS services that support IPv6 in the *Amazon VPC User Guide*.

For example, you can attach the following identity-based policy to an IAM role. This policy allows the user to put objects into the `amzn-s3-demo-bucket3` Amazon S3 bucket if they make the call from the specified IPv4 address range. This policy also allows an AWS service that uses Forward access sessions to perform this operation on your behalf.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "PrincipalPutObjectIfIpAddress",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket3/*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": "203.0.113.0/24"
                }
            }
        }
    ]
}
```

If you need to restrict access from networks that support both IPv4 and IPv6 addressing, you can include the IPv4 and IPv6 address or ranges of IP addresses in the IAM policy condition.

The following identity-based policy will allow the user to put objects into the `amzn-s3-demo-bucket3` Amazon S3 bucket if the user makes the call from either specified IPv4 or IPv6 address ranges. Before you include IPv6 address ranges in your IAM policy, verify that the AWS service you are working with supports IPv6. For a list of AWS services that support IPv6, see AWS services that support IPv6 in the *Amazon VPC User Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "PrincipalPutObjectIfIpAddress",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket3/*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "203.0.113.0/24",
                        "2001:DB8:1234:5678::/64"
                    ]
                }
            }
        }
    ]
}
```

If the request comes from a host that uses an Amazon VPC endpoint, then the `aws:SourceIp` key is not available. You should instead use a VPC-specific key such as aws:VpcSourceIp. For more information about using VPC endpoints, see Identity and access management for VPC endpoints and VPC endpoint services in the *AWS PrivateLink Guide*.

> **Note**
>
> When AWS services make calls to other AWS services on your behalf (service-to-service calls), certain network-specific authorization context is redacted. If your policy uses this condition key with `Deny` statements, AWS service principals might be unintentionally blocked. To allow AWS services to work properly while maintaining your security

requirements, exclude service principals from your `Deny` statements by adding the `aws:PrincipalIsAWSService` condition key with a value of `false`.

aws:SourceVpc

Use this key to check whether the request travels through the VPC that the VPC endpoint is attached to. In a policy, you can use this key to allow access to only a specific VPC. For more information, see Restricting Access to a Specific VPC in the *Amazon Simple Storage Service User Guide*.

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.
- **Data type** – String
- **Value type** – Single-valued

In a policy, you can use this key to allow or restrict access to a specific VPC.

For example, you can attach the following identity-based policy to an IAM role to deny `PutObject` to the `amzn-s3-demo-bucket3` Amazon S3 bucket, unless the request is made from the specified VPC ID or by AWS services that use forward access sessions (FAS) to make requests on behalf of the role. Unlike with aws:SourceIp, you must use aws:ViaAWSService or aws:CalledVia to allow FAS requests, because the source VPC of the initial request is not preserved.

Note

This policy does not allow any actions. Use this policy in combination with other policies that allow specific actions.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "PutObjectIfNotVPCID",
      "Effect": "Deny",
```

```
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket3/*",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

Note

AWS recommends to use `aws:SourceVpcArn` instead of `aws:SourceVpc` if `aws:SourceVpcArn` is supported by the service you are targeting. Please refer to aws:SourceVpcArn for the list of supported services.

Note

When AWS services make calls to other AWS services on your behalf (service-to-service calls), certain network-specific authorization context is redacted. If your policy uses this condition key with `Deny` statements, AWS service principals might be unintentionally blocked. To allow AWS services to work properly while maintaining your security requirements, exclude service principals from your `Deny` statements by adding the `aws:PrincipalIsAWSService` condition key with a value of `false`.

aws:SourceVpcArn

Use this key to verify the ARN of the VPC that a request passed through using a VPC endpoint. This key returns the ARN of the VPC to which the VPC endpoint is attached.

- **Availability** – This key is included in the request context for supported services when a request is made through a VPC endpoint. The key is not included for requests made through public service endpoints. The following services support this key:

- AWS App Runner (prefix: `apprunner`)
- AWS Application Discovery Service (prefix: `discovery`)
- Amazon Athena (prefix: `athena`)
- AWS Cloud Map (prefix: `servicediscovery`)
- Amazon CloudWatch Application Insights (prefix: `applicationinsights`)
- AWS CloudFormation (prefix: `cloudformation`)
- Amazon Comprehend Medical (prefix: `comprehendmedical`)
- AWS Compute Optimizer (prefix: `compute-optimizer`)
- Amazon; Elastic Container Registry (prefix: `ecr`)
- Amazon Elastic Container Service (prefix: `ecs`)
- Amazon Kinesis Analytics (prefix: `kinesisanalytics`)
- Amazon Route 53 (prefix: `route53`)
- AWS DataSync (prefix: `datasync`)
- Amazon Elastic Block Store (prefix: `ebs`)
- Amazon EventBridge Scheduler (prefix: `scheduler`)
- Amazon Data Firehose (prefix: `firehose`)
- AWS HealthImaging (prefix: `medical-imaging`)
- AWS HealthLake (prefix: `healthlake`)
- AWS HealthOmics (prefix: `omics`)
- AWS Identity and Access Management (except for the `iam:PassRole` action) (prefix: `iam`)
- AWS IoT FleetWise (prefix: `iotfleetwise`)
- AWS IoT Wireless (prefix: `iotwireless`)
- AWS Key Management Service (prefix: `kms`)
- AWS Lambda (prefix: `lambda`)
- AWS Payment Cryptography (prefix: `payment-cryptography`)
- Amazon Polly (prefix: `polly`)
- AWS Private Certificate Authority (prefix: `acm-pca`)
- AWS Recycle Bin (prefix: `rbin`)
- Amazon Rekognition (prefix: `rekognition`)
- Service Quotas (prefix: `servicequotas`)

- Amazon Simple Storage Service (prefix: `s3`)
- AWS Storage Gateway (prefix: `storagegateway`)
- AWS Systems Manager Incident Manager Contacts (prefix: `ssm-contacts`)
- Amazon Textract (prefix: `textract`)
- Amazon Transcribe (prefix: `transcribe`)
- AWS Transfer Family (prefix: `transfer`)

- **Data type** – ARN

  AWS recommends that you use ARN operators instead of string operators when comparing ARNs.
- **Value type** – Single-valued
- **Example value** – `arn:aws:ec2:us-east-1:123456789012:vpc/vpc-0e9801d129EXAMPLE`

The following is an example of a bucket policy that denies access to `amzn-s3-demo-bucket` and its objects from anywhere outside VPC `vpc-1a2b3c4d`.

```
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Sid": "Access-to-specific-VPC-only",
       "Principal": "*",
       "Action": "s3:*",
       "Effect": "Deny",
       "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket",
                    "arn:aws:s3:::amzn-s3-demo-bucket/*"],
       "Condition": {
         "ArnNotEquals": {
           "aws:SourceVpcArn": "arn:aws:ec2:us-east-1:*:vpc/vpc-1a2b3c4d"
         }
       }
     }
   ]
}
```

> **Note**
>
> When AWS services make calls to other AWS services on your behalf (service-to-service calls), certain network-specific authorization context is redacted. If your policy uses this

condition key with `Deny` statements, AWS service principals might be unintentionally blocked. To allow AWS services to work properly while maintaining your security requirements, exclude service principals from your `Deny` statements by adding the `aws:PrincipalIsAWSService` condition key with a value of `false`.

aws:SourceVpce

Use this key to compare the VPC endpoint identifier of the request with the endpoint ID that you specify in the policy.

- **Availability** – This key is included in the request context only if the requester uses a VPC endpoint to make the request.
- **Data type** – String
- **Value type** – Single-valued

In a policy, you can use this key to restrict access to a specific VPC endpoint. For more information, see Restricting access to a specific VPC in the *Amazon Simple Storage Service User Guide*. Similarly to using aws:SourceVpc, you must use aws:ViaAWSService or aws:CalledVia to allow requests made by AWS services using forward access sessions (FAS). This is because the source VPC endpoint of the initial request is not preserved.

Note

When AWS services make calls to other AWS services on your behalf (service-to-service calls), certain network-specific authorization context is redacted. If your policy uses this condition key with `Deny` statements, AWS service principals might be unintentionally blocked. To allow AWS services to work properly while maintaining your security requirements, exclude service principals from your `Deny` statements by adding the `aws:PrincipalIsAWSService` condition key with a value of `false`.

aws:VpceAccount

Use this key to compare the AWS account ID that owns the VPC endpoint through which the request was made with the account ID that you specify in the policy. This condition key helps you establish network perimeter controls by ensuring requests come through VPC endpoints owned by specific accounts.

- **Availability** – This key is included in the request context when a request is made through a VPC endpoint. The key is not included for requests made through public service endpoints.

  The following services support this key:

  - AWS App Runner (prefix: `apprunner`)
  - AWS Application Discovery Service (prefix: `discovery`)
  - Amazon Athena (prefix: `athena`)
  - AWS Cloud Map (prefix: `servicediscovery`)
  - Amazon CloudWatch Application Insights (prefix: `applicationinsights`)
  - AWS CloudFormation (prefix: `cloudformation`)
  - Amazon Comprehend Medical (prefix: `comprehendmedical`)
  - AWS Compute Optimizer (prefix: `compute-optimizer`)
  - Amazon; Elastic Container Registry (prefix: `ecr`)
  - Amazon Elastic Container Service (prefix: `ecs`)
  - Amazon Kinesis Analytics (prefix: `kinesisanalytics`)
  - Amazon Route 53 (prefix: `route53`)
  - AWS DataSync (prefix: `datasync`)
  - Amazon DynamoDB (prefix: `dynamodb`)
  - Amazon Elastic Block Store (prefix: `ebs`)
  - Amazon EventBridge Scheduler (prefix: `scheduler`)
  - Amazon Data Firehose (prefix: `firehose`)
  - Amazon GameLift (prefix: `gamelift`)
  - AWS HealthImaging (prefix: `medical-imaging`)
  - AWS HealthLake (prefix: `healthlake`)
  - AWS HealthOmics (prefix: `omics`)
  - AWS Identity and Access Management (except for the `iam:PassRole` action) (prefix: `iam`)
  - AWS IoT FleetWise (prefix: `iotfleetwise`)
  - AWS IoT Wireless (prefix: `iotwireless`)
  - AWS Key Management Service (prefix: `kms`)
  - AWS Lambda (prefix: `lambda`)
  - AWS Payment Cryptography (prefix: `payment-cryptography`)

- Amazon Polly (prefix: polly)
- AWS Private Certificate Authority (prefix: acm-pca)
- AWS Recycle Bin (prefix: rbin)
- Amazon Rekognition (prefix: rekognition)
- Service Quotas (prefix: servicequotas)
- Amazon Simple Storage Service (prefix: s3)
- AWS Shield (prefix: shield)
- AWS Storage Gateway (prefix: storagegateway)
- AWS Systems Manager Incident Manager Contacts (prefix: ssm-contacts)
- Amazon Textract (prefix: textract)
- Amazon Transcribe (prefix: transcribe)
- AWS Transfer Family (prefix: transfer)

- **Data type** – String
- **Value type** – Single-valued
- **Example value** – 123456789012

You can use this condition key to restrict access to resources so that requests must come through VPC endpoints owned by your account. The following Amazon S3 bucket policy example allows access when the request comes through a VPC endpoint owned by the specified account:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessToSpecificVpceAccountOnly",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/RoleName"
            },
            "Action": "s3:GetObject",
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1/*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:VpceAccount": "111122223333"
                }
            }
        }
    ]
}
```

> **Note**
>
> This condition key is currently supported for a select set of AWS services. Using this key with unsupported services can lead to unintended authorization results. Always scope the condition key to supported services in your policies.

Some AWS services access your resources from their networks when they act on your behalf. If you use such services, you will need to edit the above policy example to allow AWS services access your resources from outside your network. For more information on access patterns that need to be accounted for when enforcing access controls based on the request origin, see Establish permissions guardrails using data perimeters.

aws:VpceOrgID

Use this key to compare the identifier of the organization in AWS Organizations that owns the VPC endpoint from which the request was made with the identifier that you specify in the policy. This condition key provides the most scalable approach to network perimeter controls, automatically including all VPC endpoints owned by accounts within your organization.

- **Availability** – This key is included in the request context when a request is made through a VPC endpoint and the VPC endpoint owner account is a member of an AWS organization. The key is not included for requests made through other network paths or when the VPC endpoint owner account is not part of an organization.

  The following services support this key:

  - AWS App Runner (prefix: `apprunner`)
  - AWS Application Discovery Service (prefix: `discovery`)
  - Amazon Athena (prefix: `athena`)
  - AWS B2B Data Interchange (prefix: `b2bi`)

- AWS Cloud Map (prefix: servicediscovery)
- AWS CloudHSM (prefix: cloudhsmv2)
- Amazon CloudWatch Application Insights (prefix: applicationinsights)
- AWS CloudFormation (prefix: cloudformation)
- Amazon Cognito (prefix: cognito-identity)
- Amazon Comprehend Medical (prefix: comprehendmedical)
- AWS Compute Optimizer (prefix: compute-optimizer)
- AWS Database Migration Service (prefix: dms)
- AWS Directory Service Data (prefix: ds-data)
- Amazon DynamoDB (prefix: dynamodb)
- Amazon; Elastic Container Registry (prefix: ecr)
- Amazon Elastic Container Service (prefix: ecs)
- Amazon Kinesis Data Streams (prefix: kinesis)
- Amazon Kinesis Analytics (prefix: kinesisanalytics)
- Amazon Q Business (prefix: qbusiness)
- Amazon Route 53 (prefix: route53)
- AWS DataSync (prefix: datasync)
- Amazon Elastic Block Store (prefix: ebs)
- Amazon EventBridge Scheduler (prefix: scheduler)
- Amazon Data Firehose (prefix: firehose)
- Amazon GameLift (prefix: gamelift)
- AWS HealthImaging (prefix: medical-imaging)
- AWS HealthLake (prefix: healthlake)
- AWS HealthOmics (prefix: omics)
- AWS Identity and Access Management (except for the `iam:PassRole` action) (prefix: iam)
- AWS Identity Store (prefix: identitystore)
- AWS IoT FleetWise (prefix: iotfleetwise)
- AWS IoT TwinMaker (prefix: iottwinmaker)
- AWS IoT Wireless (prefix: iotwireless)
- Amazon Keyspaces (for Apache Cassandra) (prefix: cassandra)

- AWS Key Management Service (prefix: `kms`)
- AWS Lambda (prefix: `lambda`)
- AWS Network Firewall (prefix: `network-firewall`)
- AWS Payment Cryptography (prefix: `payment-cryptography`)
- Amazon Pinpoint SMS and Voice Service (prefix: `sms-voice`)
- Amazon Polly (prefix: `polly`)
- AWS Price List (prefix: `pricing`)
- AWS Private Certificate Authority (prefix: `acm-pca`)
- AWS Recycle Bin (prefix: `rbin`)
- Amazon Rekognition (prefix: `rekognition`)
- Service Quotas (prefix: `servicequotas`)
- Amazon Simple Email Service (prefix: `ses`)
- AWS Shield (prefix: `shield`)
- Amazon Simple Storage Service (prefix: `s3`)
- Amazon Simple Queue Service (prefix: `sqs`)
- AWS Storage Gateway (prefix: `storagegateway`)
- AWS Systems Manager Incident Manager Contacts (prefix: `ssm-contacts`)
- Amazon Textract (prefix: `textract`)
- Amazon Transcribe (prefix: `transcribe`)
- AWS Transfer Family (prefix: `transfer`)
- Amazon WorkMail (prefix: `workmail`)

- **Data type** – String
- **Value type** – Single-valued
- **Example values** – `o-a1b2c3d4e5`

The following resource control policy example denies access to your Amazon S3 and AWS Key Management Service resources unless the request comes through VPC endpoints owned by the specified organization or from networks of AWS services that act on your behalf. Some organizations might need to further edit this policy to meet the needs of their organization, for example, allow third-party partner access. For more information on access patterns that need

to be accounted for when enforcing access controls based on the request origin, see Establish permissions guardrails using data perimeters.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceNetworkPerimeterVpceOrgID",
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "s3:*",
        "kms:*"
      ],
      "Resource": "*",
      "Condition": {
        "BoolIfExists": {
          "aws:PrincipalIsAWSService": "false",
          "aws:ViaAWSService": "false"
        },
        "StringNotEqualsIfExists": {
            "aws:VpceOrgID": "o-abcdef0123",
            "aws:PrincipalTag/network-perimeter-exception": "true"
        }
      }
    }
  ]
}
```

Note

This condition key is currently supported for a select set of AWS services. Using this key with unsupported services can lead to unintended authorization results. Always scope the condition key to supported services in your policies.

Note

When AWS services make calls to other AWS services on your behalf (service-to-service calls), certain network-specific authorization context is redacted. If your policy uses this condition key with `Deny` statements, AWS service principals might be unintentionally blocked. To allow AWS services to work properly while maintaining your security requirements, exclude service principals from your `Deny` statements by adding the `aws:PrincipalIsAWSService` condition key with a value of `false`.

aws:VpceOrgPaths

Use this key to compare the AWS Organizations path for the VPC endpoint from which the request was made with the path that you specify in the policy. This condition key enables you to implement network perimeter controls at the organizational unit (OU) level, automatically scaling with your VPC endpoint usage as you add new endpoints within the specified OUs.

- **Availability** – This key is included in the request context when a request is made through a VPC endpoint and the VPC endpoint owner account is a member of an organization. The key is not included for requests made through other network paths or when the VPC endpoint owner account is not part of an organization.

  The following services support this key:

  - AWS App Runner (prefix: `apprunner`)
  - AWS Application Discovery Service (prefix: `discovery`)
  - Amazon Athena (prefix: `athena`)
  - AWS Cloud Map (prefix: `servicediscovery`)
  - Amazon CloudWatch Application Insights (prefix: `applicationinsights`)
  - AWS CloudFormation (prefix: `cloudformation`)
  - Amazon Comprehend Medical (prefix: `comprehendmedical`)
  - AWS Compute Optimizer (prefix: `compute-optimizer`)
  - Amazon; Elastic Container Registry (prefix: `ecr`)
  - Amazon Elastic Container Service (prefix: `ecs`)
  - Amazon Kinesis Analytics (prefix: `kinesisanalytics`)
  - Amazon Route 53 (prefix: `route53`)

- AWS DataSync (prefix: `datasync`)
- Amazon DynamoDB (prefix: `dynamodb`)
- Amazon Elastic Block Store (prefix: `ebs`)
- Amazon EventBridge Scheduler (prefix: `scheduler`)
- Amazon Data Firehose (prefix: `firehose`)
- Amazon GameLift (prefix: `gamelift`)
- AWS HealthImaging (prefix: `medical-imaging`)
- AWS HealthLake (prefix: `healthlake`)
- AWS HealthOmics (prefix: `omics`)
- AWS Identity and Access Management (except for the `iam:PassRole` action) (prefix: `iam`)
- AWS IoT FleetWise (prefix: `iotfleetwise`)
- AWS IoT Wireless (prefix: `iotwireless`)
- AWS Key Management Service (prefix: `kms`)
- AWS Lambda (prefix: `lambda`)
- AWS Payment Cryptography (prefix: `payment-cryptography`)
- Amazon Polly (prefix: `polly`)
- AWS Private Certificate Authority (prefix: `acm-pca`)
- AWS Recycle Bin (prefix: `rbin`)
- Amazon Rekognition (prefix: `rekognition`)
- Service Quotas (prefix: `servicequotas`)
- AWS Secrets Manager (prefix: `secretsmanager`)
- AWS Security Token Service (prefix: `sts`)
- Amazon Simple Storage Service (prefix: `s3`)
- AWS Shield (prefix: `shield`)
- AWS Storage Gateway (prefix: `storagegateway`)
- AWS Systems Manager Incident Manager Contacts (prefix: `ssm-contacts`)
- Amazon Textract (prefix: `textract`)
- Amazon Transcribe (prefix: `transcribe`)
- AWS Transfer Family (prefix: `transfer`)

- **Data type** – String (list)

- **Value type** – Multivalued
- **Example values** – `o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/`

Since `aws:VpceOrgPaths` is a multivalued condition key, you must use the `ForAnyValue` or `ForAllValues` set operators with string condition operators for this key. The following Amazon S3 bucket policy example allows access only when requests come through VPC endpoints owned by accounts in specific organizational units:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessFromSpecificOrgPaths",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/RoleName"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket1/*",
      "Condition": {
        "ForAnyValue:StringLike": {
          "aws:VpceOrgPaths": [
            "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/*",
            "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-33333333/*"
          ]
        }
      }
    }
  ]
}
```

Note

This condition key is currently supported for a select set of AWS services. Using this key with unsupported services can lead to unintended authorization results. Always scope the condition key to supported services in your policies.

Some AWS services access your resources from their networks when they act on your behalf. If you use such services, you will need to edit the above policy example to allow AWS services access your resources from outside your network. For more information on access patterns that need to be accounted for when enforcing access controls based on the request origin, see Establish permissions guardrails using data perimeters.

aws:VpcSourceIp

Use this key to compare the IP address from which a request was made with the IP address that you specify in the policy. In a policy, the key matches only if the request originates from the specified IP address and it goes through a VPC endpoint.

- **Availability** – This key is included in the request context only if the request is made using a VPC endpoint.
- **Data type** – IP address
- **Value type** – Single-valued

For more information, see Control access to VPC endpoints using endpoint policies in the *Amazon VPC User Guide*. Similarly to using aws:SourceVpc, you must use aws:ViaAWSService or aws:CalledVia to allow requests made by AWS services using forward access sessions (FAS). This is because the source IP of the initial request made using a VPC endpoint is not preserved in FAS requests.

Note

`aws:VpcSourceIp` supports both IPv4 and IPv6 address or range of IP addresses. For a list of AWS services that support IPv6, see AWS services that support IPv6 in the *Amazon VPC User Guide*.
Always use the `aws:VpcSourceIp` condition key with the `aws:SourceVpc`, `aws:SourceVpce`, or `aws:SourceVpcArn` condition keys. If you don't, a policy might permit API calls from an unexpected VPC that uses the same or overlapping IP CIDR. This can happen because IP CIDRs from two unrelated VPCs can be the same or overlap.

Note

When AWS services make calls to other AWS services on your behalf (service-to-service calls), certain network-specific authorization context is redacted. If your policy uses this

condition key with `Deny` statements, AWS service principals might be unintentionally blocked. To allow AWS services to work properly while maintaining your security requirements, exclude service principals from your `Deny` statements by adding the `aws:PrincipalIsAWSService` condition key with a value of `false`.

Properties of the resource

Use the following condition keys to compare details about the resource that is the target of the request with the resource properties that you specify in the policy.

aws:ResourceAccount

Use this key to compare the requested resource owner's AWS account ID with the resource account in the policy. You can then allow or deny access to that resource based on the account that owns the resource.

- **Availability** – This key is always included in the request context for most service actions. The following actions don't support this key:
  - AWS Audit Manager
    - `auditmanager:UpdateAssessmentFrameworkShare`
  - Amazon Detective
    - `detective:AcceptInvitation`
  - AWS Directory Service
    - `ds:AcceptSharedDirectory`
  - Amazon Elastic Block Store – All actions
  - Amazon EC2
    - `ec2:AcceptTransitGatewayPeeringAttachment`
    - `ec2:AcceptVpcEndpointConnections`
    - `ec2:AcceptVpcPeeringConnection`
    - `ec2:CreateTransitGatewayPeeringAttachment`
    - `ec2:CreateVpcEndpoint`
    - `ec2:CreateVpcPeeringConnection`
  - Amazon EventBridge

    - `events:PutEvents` – EventBridge `PutEvents` calls on an event bus in another account, if that event bus was configured as a cross-account EventBridge target before March 2, 2023. For more information, see Grant permissions to allow events from other AWS accounts in the *Amazon EventBridge User Guide*.
- Amazon GuardDuty
    - `guardduty:AcceptAdministratorInvitation`
- Amazon Macie
    - `macie2:AcceptInvitation`
- Amazon OpenSearch Service
    - `es:AcceptInboundConnection`
- Amazon Route 53
    - `route53:AssociateVpcWithHostedZone`
    - `route53:CreateVPCAssociationAuthorization`
- AWS Security Hub CSPM
    - `securityhub:AcceptAdministratorInvitation`
- **Data type** – String
- **Value type** – Single-valued

> **Note**
>
> For additional considerations for the above unsupported actions, see the Data Perimeter Policy Examples repository.

This key is equal to the AWS account ID for the account with the resources evaluated in the request.

For most resources in your account, the ARN contains the owner account ID for that resource. For certain resources, such as Amazon S3 buckets, the resource ARN does not include the account ID. The following two examples show the difference between a resource with an account ID in the ARN, and an Amazon S3 ARN without an account ID:

- `arn:aws:iam::123456789012:role/AWSExampleRole` – IAM role created and owned within the account 123456789012.

- `arn:aws:s3:::amzn-s3-demo-bucket2` – Amazon S3 bucket created and owned within the account 111122223333, not displayed in the ARN.

Use the AWS console, or API, or CLI, to find all of your resources and corresponding ARNs.

You write a policy that denies permissions to resources based on the resource owner's account ID. For example, the following identity-based policy denies access to the *specified resource* if the resource does not belong to the *specified account*.

To use this policy, replace the *italicized placeholder text* with your account information.

> **Important**
>
> This policy does not allow any actions. Instead, it uses the Deny effect which explicitly denies access to all of the resources listed in the statement that do not belong to the listed account. Use this policy in combination with other policies that allow access to specific resources.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyInteractionWithResourcesNotInSpecificAccount",
      "Action": "service:*",
      "Effect": "Deny",
      "Resource": [
        "arn:aws:service:us-east-1:111122223333:*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "111122223333"
          ]
        }
      }
    }
  ]
```

```
}
```

This policy denies access to all resources for a specific AWS service unless the specified AWS account owns the resource.

> **Note**
>
> Some AWS services require access to AWS owned resources that are hosted in another AWS account. Using `aws:ResourceAccount` in your identity-based policies might impact your identity's ability to access these resources.

Certain AWS services, such as AWS Data Exchange, rely on access to resources outside of your AWS accounts for normal operations. If you use the element `aws:ResourceAccount` in your policies, include additional statements to create exemptions for those services. The example policy AWS: Deny access to Amazon S3 resources outside your account except AWS Data Exchange demonstrates how to deny access based on the resource account while defining exceptions for service-owned resources.

Use this policy example as a template for creating your own custom policies. Refer to your service documentation for more information.

aws:ResourceOrgPaths

Use this key to compare the AWS Organizations path for the accessed resource to the path in the policy. In a policy, this condition key ensures that the resource belongs to an account member within the specified organization root or organizational units (OUs) in AWS Organizations. An AWS Organizations path is a text representation of the structure of an Organizations entity. For more information about using and understanding paths, see Understand the AWS Organizations entity path

- **Availability** – This key is included in the request context only if the account that owns the resource is a member of an organization. This global condition key does not support the following actions:
  - AWS Audit Manager
    - `auditmanager:UpdateAssessmentFrameworkShare`
  - Amazon Detective

    - `detective:AcceptInvitation`
- AWS Directory Service
    - `ds:AcceptSharedDirectory`
- Amazon Elastic Block Store – All actions
- Amazon EC2
    - `ec2:AcceptTransitGatewayPeeringAttachment`
    - `ec2:AcceptVpcEndpointConnections`
    - `ec2:AcceptVpcPeeringConnection`
    - `ec2:CreateTransitGatewayPeeringAttachment`
    - `ec2:CreateVpcEndpoint`
    - `ec2:CreateVpcPeeringConnection`
- Amazon EventBridge
    - `events:PutEvents` – EventBridge `PutEvents` calls on an event bus in another account, if that event bus was configured as a cross-account EventBridge target before March 2, 2023. For more information, see Grant permissions to allow events from other AWS accounts in the *Amazon EventBridge User Guide*.
- Amazon GuardDuty
    - `guardduty:AcceptAdministratorInvitation`
- Amazon Macie
    - `macie2:AcceptInvitation`
- Amazon OpenSearch Service
    - `es:AcceptInboundConnection`
- Amazon Route 53
    - `route53:AssociateVpcWithHostedZone`
    - `route53:CreateVPCAssociationAuthorization`
- AWS Security Hub CSPM
    - `securityhub:AcceptAdministratorInvitation`
- **Data type** – String (list)
- **Value type** – Multivalued

> **Note**
>
> For additional considerations for the above unsupported actions, see the Data Perimeter Policy Examples repository.

`aws:ResourceOrgPaths` is a multivalued condition key. Multivalued keys can have multiple values in the request context. You must use the `ForAnyValue` or `ForAllValues` set operators with string condition operators for this key. For more information about multivalued condition keys, see Set operators for multivalued context keys.

For example, the following condition returns `True` for resources that belong to the organization `o-a1b2c3d4e5`. When you include a wildcard, you must use the StringLike condition operator.

```
"Condition": {
      "ForAnyValue:StringLike": {
            "aws:ResourceOrgPaths":["o-a1b2c3d4e5/*"]
   }
}
```

The following condition returns `True` for resources with the OU ID `ou-ab12-11111111`. It will match resources owned by accounts attached to the OU ou-ab12-11111111 or any of the child OUs.

```
"Condition": { "ForAnyValue:StringLike" : {
     "aws:ResourceOrgPaths":["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/*"]
}}
```

The following condition returns `True` for resources owned by accounts attached directly to the OU ID `ou-ab12-22222222`, but not the child OUs. The following example uses the StringEquals condition operator to specify the exact match requirement for the OU ID and not a wildcard match.

```
"Condition": { "ForAnyValue:StringEquals" : {
     "aws:ResourceOrgPaths":["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-ab12-22222222/"]
}}
```

> **Note**
>
> Some AWS services require access to AWS owned resources that are hosted in another AWS account. Using `aws:ResourceOrgPaths` in your identity-based policies might impact your identity's ability to access these resources.

Certain AWS services, such as AWS Data Exchange, rely on access to resources outside of your AWS accounts for normal operations. If you use the `aws:ResourceOrgPaths` key in your policies, include additional statements to create exemptions for those services. The example policy AWS: Deny access to Amazon S3 resources outside your account except AWS Data Exchange demonstrates how to deny access based on the resource account while defining exceptions for service-owned resources. You can create a similar policy to restrict access to resources within an organizational unit (OU) using the `aws:ResourceOrgPaths` key, while accounting for service-owned resources.

Use this policy example as a template for creating your own custom policies. Refer to your service documentation for more information.

aws:ResourceOrgID

Use this key to compare the identifier of the organization in AWS Organizations to which the requested resource belongs with the identifier specified in the policy.

- **Availability** – This key is included in the request context only if the account that owns the resource is a member of an organization. This global condition key does not support the following actions:
  - AWS Audit Manager
    - `auditmanager:UpdateAssessmentFrameworkShare`
  - Amazon Detective
    - `detective:AcceptInvitation`
  - AWS Directory Service
    - `ds:AcceptSharedDirectory`
  - Amazon Elastic Block Store – All actions
  - Amazon EC2
    - `ec2:AcceptTransitGatewayPeeringAttachment`

    - `ec2:AcceptVpcEndpointConnections`
    - `ec2:AcceptVpcPeeringConnection`
    - `ec2:CreateTransitGatewayPeeringAttachment`
    - `ec2:CreateVpcEndpoint`
    - `ec2:CreateVpcPeeringConnection`
- Amazon EventBridge
    - `events:PutEvents` – EventBridge `PutEvents` calls on an event bus in another account, if that event bus was configured as a cross-account EventBridge target before March 2, 2023. For more information, see Grant permissions to allow events from other AWS accounts in the *Amazon EventBridge User Guide*.
- Amazon GuardDuty
    - `guardduty:AcceptAdministratorInvitation`
- Amazon Macie
    - `macie2:AcceptInvitation`
- Amazon OpenSearch Service
    - `es:AcceptInboundConnection`
- Amazon Route 53
    - `route53:AssociateVpcWithHostedZone`
    - `route53:CreateVPCAssociationAuthorization`
- AWS Security Hub CSPM
    - `securityhub:AcceptAdministratorInvitation`
- **Data type** – String
- **Value type** – Single-valued

> **Note**
>
> For additional considerations for the above unsupported actions, see the Data Perimeter Policy Examples repository.

This global key returns the resource organization ID for a given request. It allows you to create rules that apply to all resources in an organization that are specified in the `Resource` element

of an identity-based policy. You can specify the organization ID in the condition element. When you add and remove accounts, policies that include the `aws:ResourceOrgID` key automatically include the correct accounts and you don't have to manually update it.

For example, the following policy prevents the principal from adding objects to the `policy-genius-dev` resource unless the Amazon S3 resource belongs to the same organization as the principal making the request.

Important

This policy does not allow any actions. Instead, it uses the `Deny` effect which explicitly denies access to all of the resources listed in the statement that do not belong to the listed account. Use this policy in combination with other policies that allow access to specific resources.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "DenyPutObjectToS3ResourcesOutsideMyOrganization",
        "Effect": "Deny",
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::policy-genius-dev/*",
        "Condition": {
            "StringNotEquals": {
                "aws:ResourceOrgID": "${aws:PrincipalOrgID}"
            }
        }
    }
}
```

Note

Some AWS services require access to AWS owned resources that are hosted in another AWS account. Using `aws:ResourceOrgID` in your identity-based policies might impact your identity's ability to access these resources.

Certain AWS services, such as AWS Data Exchange, rely on access to resources outside of your AWS accounts for normal operations. If you use the `aws:ResourceOrgID` key in your policies, include additional statements to create exemptions for those services. The example policy AWS: Deny access to Amazon S3 resources outside your account except AWS Data Exchange demonstrates how to deny access based on the resource account while defining exceptions for service-owned resources. You can create a similar policy to restrict access to resources within your organization using the `aws:ResourceOrgID` key, while accounting for service-owned resources.

Use this policy example as a template for creating your own custom policies. Refer to your service documentation for more information.

In the following video, learn more about how you might use the `aws:ResourceOrgID` condition key in a policy.

Ensure identities and networks can only be used to access trusted resources.

aws:ResourceTag/*tag-key*

Use this key to compare the tag key-value pair that you specify in the policy with the key-value pair attached to the resource. For example, you could require that access to a resource is allowed only if the resource has the attached tag key `"Dept"` with the value `"Marketing"`. For more information, see Controlling access to AWS resources.

- **Availability** – This key is included in the request context when the requested resource already has attached tags or in requests that create a resource with an attached tag. This key is returned only for resources that support authorization based on tags. There is one context key for each tag key-value pair.
- **Data type** – String
- **Value type** – Single-valued

This context key is formatted `"aws:ResourceTag/`*`tag-key`*`":"`*`tag-value`*`"` where *`tag-key`* and *`tag-value`* are a tag key and value pair. Tag keys are not case-sensitive. This means that if you specify `"aws:ResourceTag/TagKey1": "Value1"` in the condition element of your policy, then the condition matches a resource tag key named either `TagKey1` or `tagkey1`, but not both. Values in these tag key/value pairs are case-sensitive. This means that if you specify `"aws:ResourceTag/TagKey1": "Production"` in the condition element of your policy, then the condition matches a resource tag value named `Production` but it would not match `production` or `PRODUCTION`.

For examples of using the `aws:ResourceTag` key to control access to IAM resources, see Controlling access to AWS resources.

For examples of using the `aws:ResourceTag` key to control access to other AWS resources, see Controlling access to AWS resources using tags.

For a tutorial on using the `aws:ResourceTag` condition key for attribute based access control (ABAC), see IAM tutorial: Define permissions to access AWS resources based on tags.

Properties of the request

Use the following condition keys to compare details about the request itself and the contents of the request with the request properties that you specify in the policy.

aws:CalledVia

Use this key to compare the services in the policy with the services that made requests on behalf of the IAM principal (user or role). When a principal makes a request to an AWS service, that service might use the principal's credentials to make subsequent requests to other services. When the request is made using forward access sessions (FAS), this key is set with the value of the service principal. The `aws:CalledVia` key contains an ordered list of each service in the chain that made requests on the principal's behalf.

For more information, see Forward access sessions.

- **Availability** – This key is present in the request when a service that supports `aws:CalledVia` uses the credentials of an IAM principal to make a request to another service. This key is not present if the service uses a service role or service-linked role to make a call on the principal's behalf. This key is also not present when the principal makes the call directly.
- **Data type** – String (list)
- **Value type** – Multivalued

To use the `aws:CalledVia` condition key in a policy, you must provide the service principals to allow or deny AWS service requests. For example, you can use AWS CloudFormation to read and write from an Amazon DynamoDB table. DynamoDB then uses encryption supplied by AWS Key Management Service (AWS KMS).

To allow or deny access when *any* service makes a request using the principal's credentials, use the `aws:ViaAWSService` condition key. That condition key supports AWS services.

The `aws:CalledVia` key is a multivalued key. However, you can't enforce order using this key in a condition. Using the example above, **User 1** makes a request to CloudFormation, which calls DynamoDB, which calls AWS KMS. These are three separate requests. The final call to AWS KMS is performed by User 1 through CloudFormation and then DynamoDB.
Example using aws:CalledVia.

In this case, the `aws:CalledVia` key in the request context includes `cloudformation.amazonaws.com` and `dynamodb.amazonaws.com`, in that order. If you care only that the call was made through DynamoDB somewhere in the chain of requests, you can use this condition key in your policy.

For example, the following policy allows managing the AWS KMS key named `my-example-key`, but only if DynamoDB is one of the requesting services. The `ForAnyValue:StringEquals` condition operator ensures that DynamoDB is one of the calling services. If the principal makes the call to AWS KMS directly, the condition returns `false` and the request is not allowed by this policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "KmsActionsIfCalledViaDynamodb",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/my-example-key",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": [
                        "dynamodb.amazonaws.com"
                    ]
                }
            }
        }
    ]
```

```
}
```

If you want to enforce which service makes the first or last call in the chain, you can use the aws:CalledViaFirst and aws:CalledViaLast keys. For example, the following policy allows managing the key named my-example-key in AWS KMS. These AWS KMS operations are allowed only if multiple requests were included in the chain. The first request must be made through CloudFormation and the last through DynamoDB. If other services make requests in the middle of the chain, the operation is still allowed.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "KmsActionsIfCalledViaChain",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/my-example-key",
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaFirst": "cloudformation.amazonaws.com",
                    "aws:CalledViaLast": "dynamodb.amazonaws.com"
                }
            }
        }
    ]
}
```

The aws:CalledViaFirst and aws:CalledViaLast keys are present in the request when a service uses an IAM principal's credentials to call another service. They indicate the first and last services that made calls in the chain of requests. For example, assume that CloudFormation calls another service named X Service, which calls DynamoDB, which then calls AWS KMS. The final

call to AWS KMS is performed by `User 1` through CloudFormation, then `X Service`, and then DynamoDB. It was first called through CloudFormation and last called through DynamoDB. Example using aws:CalledViaFirst and aws:CalledViaLast.

aws:CalledViaFirst

Use this key to compare the services in the policy with the ***first service*** that made a request on behalf of the IAM principal (user or role). For more information, see `aws:CalledVia`.

- **Availability** – This key is present in the request when a service uses the credentials of an IAM principal to make at least one other request to a different service. This key is not present if the service uses a service role or service-linked role to make a call on the principal's behalf. This key is also not present when the principal makes the call directly.
- **Data type** – String
- **Value type** – Single-valued

aws:CalledViaLast

Use this key to compare the services in the policy with the *last service* that made a request on behalf of the IAM principal (user or role). For more information, see `aws:CalledVia`.

- **Availability** – This key is present in the request when a service uses the credentials of an IAM principal to make at least one other request to a different service. This key is not present if the service uses a service role or service-linked role to make a call on the principal's behalf. This key is also not present when the principal makes the call directly.
- **Data type** – String
- **Value type** – Single-valued

aws:ViaAWSService

Use this key to check whether an AWS service makes a request to another service on your behalf using forward access sessions (FAS).

The request context key returns `true` when a service uses forward access sessions to make a request on behalf of the original IAM principal. The request context key also returns `false` when the principal makes the call directly.

- **Availability** – This key is always included in the request context.

- **Data type** – Boolean
- **Value type** – Single-valued

aws:ViaCustomerDomain

Use this key to check whether a principal made a request by using credentials obtained from Temporary Access Delegation (TAD). When a partner or vendor obtains temporary credentials through a TAD token exchange, we set this key to the account identifier of the requesting partner.

Use this condition key in AWS Identity and Access Management (IAM) policies to scope access controls specifically to sessions that temporary access delegation established, without affecting your own sessions. For example, you can use this key to revoke only delegation-based sessions while preserving direct console access.

- **Availability** – This key is present in the request context when a principal uses credentials obtained through a TAD token exchange.
- **Data type** – String
- **Value type** – Single-valued

The following example shows a deny policy for sessions obtained through temporary access delegation from a specific partner account. The policy applies only to sessions issued before a specified time. This example uses a wildcard (*) for `Action` and `Resource` in the `Deny` statement, which is the standard pattern for revoking all access from a specific session. To revoke only specific actions, replace the wildcard with the appropriate action list.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "DateLessThan": {
                    "aws:TokenIssueTime": "2026-07-30T01:00:37.819Z"
                },
                "StringEquals": {
                    "aws:ViaCustomerDomain": "123456789012"
                }
```

```
            }
        }
    ]
}
```

aws:CalledViaAWSMCP

Use this key to compare the services in the policy with the AWS MCP services that made requests on behalf of the IAM principal (user or role). When a principal makes a request to an AWS MCP service, that service uses the principal's credentials to make subsequent requests to other services. When the request is made using an AWS MCP service, this key is set with the value of the service principal. The `aws:CalledViaAWSMCP` key contains the service principal name of the MCP service that made requests on the principal's behalf.

- **Availability** – This key is present in the request when an AWS MCP service uses the credentials of an IAM principal to make a request to an AWS service. This key is also not present when the principal makes the call directly.
- **Data type** – String
- **Value type** – Single-valued

You can use this condition key to allow or deny access based on which specific MCP server initiated the request. For example, the following policy denies sensitive delete operations when they are initiated through a specific MCP server:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenySensitiveActionsViaSpecificMCP",
            "Effect": "Deny",
            "Action": [
                "s3:DeleteBucket",
                "s3:DeleteObject",
                "dynamodb:DeleteTable"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaAWSMCP": "aws-mcp.amazonaws.com"
                }
```

```
            }
        }
    ]
}
```

aws:ViaAWSMCPService

Use this key to check whether an AWS MCP service makes a request to another AWS service on your behalf using forward access sessions (FAS). The request context key returns `true` when an AWS MCP service forwards a request to an AWS service on behalf of the original IAM principal. The request context key also returns `false` when the principal makes the call directly.

- **Availability** – This key is included in the request context when an AWS MCP server makes a request to a downstream AWS service on behalf of an IAM principal.
- **Data type** – Boolean
- **Value type** – Single-valued

You can use this key to restrict specific actions when they come through MCP servers. For example, the following policy denies sensitive delete operations when initiated through any AWS MCP server:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenySensitiveActionsViaMCP",
            "Effect": "Deny",
            "Action": [
                "s3:DeleteBucket",
                "s3:DeleteObject",
                "dynamodb:DeleteTable"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "aws:ViaAWSMCPService": "true"
                }
            }
        }
    ]
```

```
}
```

aws:CurrentTime

Use this key to compare the date and time of the request with the date and time that you specify in the policy. To view an example policy that uses this condition key, see AWS: Allows access based on date and time.

- **Availability** – This key is always included in the request context.
- **Data type** – Date
- **Value type** – Single-valued

aws:EpochTime

Use this key to compare the date and time of the request in epoch or Unix time with the value that you specify in the policy. This key also accepts the number of seconds since January 1, 1970.

- **Availability** – This key is always included in the request context.
- **Data type** – Date, Numeric
- **Value type** – Single-valued

aws:referer

Use this key to compare who referred the request in the client browser with the referer that you specify in the policy. The `aws:referer` request context value is provided by the caller in an HTTP header. The `Referer` header is included in a web browser request when you select a link on a web page. The `Referer` header contains the URL of the web page where the link was selected.

- **Availability** – This key is included in the request context only if the request to the AWS resource was invoked by linking from a web page URL in the browser. This key is not included for programmatic requests because it doesn't use a browser link to access the AWS resource.
- **Data type** – String
- **Value type** – Single-valued

For example, you can access an Amazon S3 object directly using a URL or using direct API invocation. For more information, see Amazon S3 API operations directly using a web browser. When you access an Amazon S3 object from a URL that exists in a webpage, the URL of the source

web page is in used in `aws:referer`. When you access an Amazon S3 object by typing the URL into your browser, `aws:referer` is not present. When you invoke the API directly, `aws:referer` is also not present. You can use the `aws:referer` condition key in a policy to allow requests made from a specific referer, such as a link on a web page in your company's domain.

> **Warning**
>
> This key should be used carefully. It is dangerous to include a publicly known referer header value. Unauthorized parties can use modified or custom browsers to provide any `aws:referer` value that they choose. As a result, `aws:referer` should not be used to prevent unauthorized parties from making direct AWS requests. It is offered only to allow customers to protect their digital content, such as content stored in Amazon S3, from being referenced on unauthorized third-party sites.

aws:RequestedRegion

Use this key to compare the AWS Region that was called in the request with the Region that you specify in the policy. You can use this global condition key to control which Regions can be requested. To view the AWS Regions for each service, see Service endpoints and quotas in the *Amazon Web Services General Reference*.

- **Availability** – This key is always included in the request context.
- **Data type** – String
- **Value type** – Single-valued

Some global services, such as IAM, have a single endpoint. Because this endpoint is physically located in the US East (N. Virginia) Region, IAM calls are always made to the us-east-1 Region. For example, if you create a policy that denies access to all services if the requested Region is not us-west-2, then IAM calls always fail. To view an example of how to work around this, see NotAction with Deny.

> **Note**
>
> The `aws:RequestedRegion` condition key allows you to control which endpoint of a service is invoked but does not control the impact of the operation. Some services have cross-Region impacts.
> For example, Amazon S3 has API operations that extend across regions.

- You can invoke `s3:PutBucketReplication` in one Region (which is affected by the `aws:RequestedRegion` condition key), but other Regions are affected based on the replications configuration settings.
- You can invoke `s3:CreateBucket` to create a bucket in another region, and use the `s3:LocationConstraint` condition key to control the applicable regions.

You can use this context key to limit access to AWS services within a given set of Regions. For example, the following policy allows a user to view all of the Amazon EC2 instances in the AWS Management Console. However it only allows them to make changes to instances in Ireland (eu-west-1), London (eu-west-2), or Paris (eu-west-3).

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "InstanceConsoleReadOnly",
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:Export*",
                "ec2:Get*",
                "ec2:Search*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "InstanceWriteRegionRestricted",
            "Effect": "Allow",
            "Action": [
                "ec2:Associate*",
                "ec2:Import*",
                "ec2:Modify*",
                "ec2:Monitor*",
                "ec2:Reset*",
                "ec2:Run*",
                "ec2:Start*",
                "ec2:Stop*",
```

```
                "ec2:Terminate*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": [
                        "eu-west-1",
                        "eu-west-2",
                        "eu-west-3"
                    ]
                }
            }
        }
    ]
}
```

aws:RequestTag/*tag-key*

Use this key to compare the tag key-value pair that was passed in the request with the tag pair that you specify in the policy. For example, you could check whether the request includes the tag key `"Dept"` and that it has the value `"Accounting"`. For more information, see Controlling access during AWS requests.

- **Availability** – This key is included in the request context when tag key-value pairs are passed in the request. When multiple tags are passed in the request, there is one context key for each tag key-value pair.
- **Data type** – String
- **Value type** – Single-valued

This context key is formatted `"aws:RequestTag/`*`tag-key`*`":"`*`tag-value`*`"` where *`tag-key`* and *`tag-value`* are a tag key and value pair. Tag keys are not case-sensitive. This means that if you specify `"aws:RequestTag/TagKey1": "Value1"` in the condition element of your policy, then the condition matches a request tag key named either `TagKey1` or `tagkey1`, but not both. Values in these tag key/value pairs are case-sensitive. This means that if you specify `"aws:RequestTag/TagKey1": "Production"` in the condition element of your policy, then the condition matches a request tag value named `Production` but it would not match `production` or `PRODUCTION`.

This example shows that while the key is single-valued, you can still use multiple key-value pairs in a request if the keys are different.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:CreateTags",
    "Resource": "arn:aws:ec2::111122223333:instance/*",
    "Condition": {
      "StringEquals": {
        "aws:RequestTag/environment": [
          "preprod",
          "production"
        ],
        "aws:RequestTag/team": [
          "engineering"
        ]
      }
    }
  }
}
```

aws:TagKeys

Use this key to compare the tag keys in a request with the keys that you specify in the policy. We recommend that when you use policies to control access using tags, use the `aws:TagKeys` condition key to define what tag keys are allowed. For example policies and more information, see the section called “Controlling access based on tag keys”.

- **Availability** – This key is included in the request context if the operation supports passing tags in the request.
- **Data type** – String (list)
- **Value type** – Multivalued

This context key is formatted `"aws:TagKeys":"`*`tag-key`*`"` where *`tag-key`* is a list of tag keys without values (for example, `["Dept","Cost-Center"]`).

Because you can include multiple tag key-value pairs in a request, the request content could be a multivalued request. In this case, you must use the `ForAllValues` or `ForAnyValue` set operators. For more information, see Set operators for multivalued context keys.

Some services support tagging with resource operations, such as creating, modifying, or deleting a resource. To allow tagging and operations as a single call, you must create a policy that includes both the tagging action and the resource-modifying action. You can then use the `aws:TagKeys` condition key to enforce using specific tag keys in the request. For example, to limit tags when someone creates an Amazon EC2 snapshot, you must include the `ec2:CreateSnapshot` creation action ***and*** the `ec2:CreateTags` tagging action in the policy. To view a policy for this scenario that uses `aws:TagKeys`, see Creating a Snapshot with Tags in the *Amazon EC2 User Guide*.

aws:SecureTransport

Use this key to check whether the request was sent using TLS. The request context returns `true` or `false`. In a policy, you can allow specific actions only if the request is sent using TLS.

- **Availability** – This key is always included in the request context.
- **Data type** – Boolean
- **Value type** – Single-valued

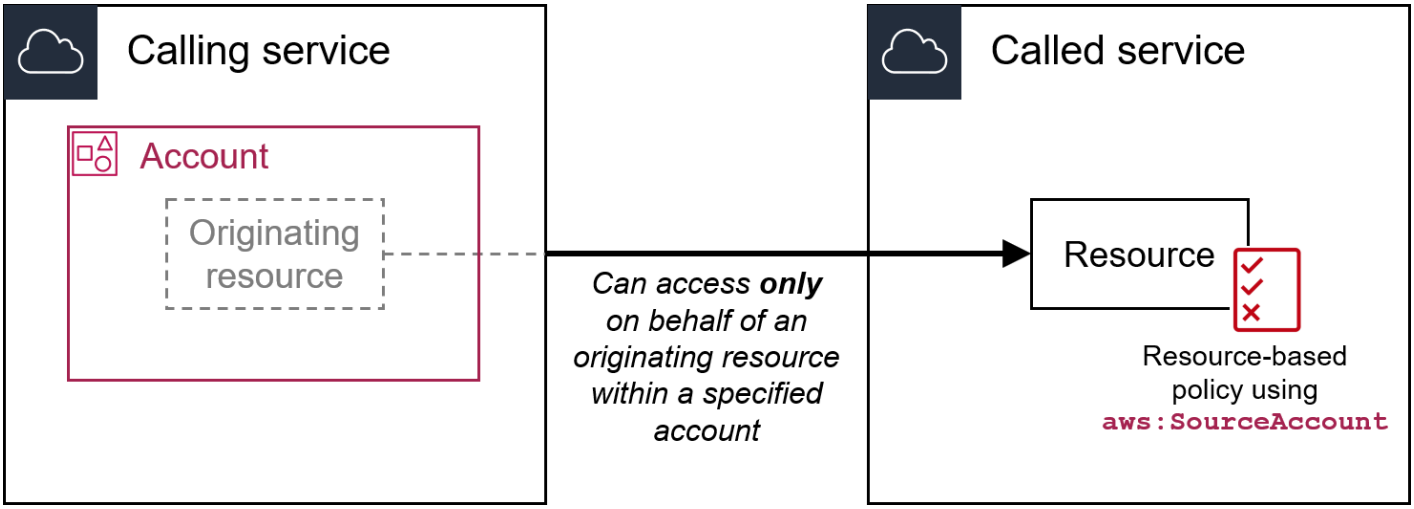
aws:SourceAccount

Use this key to compare the account ID of the resource making a service-to-service request with the account ID that you specify in the policy, but only when the request is made by an AWS service principal.

- **Availability** – This key is included in the request context only when the call to your resource is being made directly by an AWS service principal on behalf of a resource for which the configuration triggered the service-to-service request. The calling service passes the account ID of the original resource to the called service.

> **Note**
>
> This key provides a uniform mechanism for enforcing cross-service confused deputy control across AWS services. However, not all service integrations require the use of this global condition key. See the documentation of the AWS services you use for more

information about service-specific mechanisms for mitigating cross-service confused deputy risks.



- **Data type** – String
- **Value type** – Single-valued

You can use this condition key to help make sure that a calling service can access your resource only when the request originates from a specific account. For example, you can attach the following resource control policy (RCP) to deny requests by service principals against Amazon S3 buckets, unless they were triggered by a resource in the specified account. This policy applies the control only on requests by service principals (`"Bool": {"aws:PrincipalIsAWSService": "true"}`) that have the `aws:SourceAccount` key present (`"Null": {"aws:SourceAccount": "false"}`), so that service integrations that don't require the use of this key and calls by your principals aren't impacted. If the `aws:SourceAccount` key is present in the request context, the `Null` condition will evaluate to `true`, causing the `aws:SourceAccount` key to be enforced.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "RCPEnforceConfusedDeputyProtection",
      "Effect": "Deny",
      "Principal": "*",
```

```
      "Action": [
        "s3:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceAccount": "111122223333"
        },
        "Null": {
          "aws:SourceAccount": "false"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "true"
        }
      }
    }
  ]
}
```

In resource-based policies where the principal is an AWS service principal, use the key to limit permissions granted to the service. For example, when an Amazon S3 bucket is configured to send notifications to an Amazon SNS topic, the Amazon S3 service invokes the `sns:Publish` API operation for all configured events. In the topic policy that allows the `sns:Publish` operation, set the value of the condition key to the account ID of the Amazon S3 bucket.

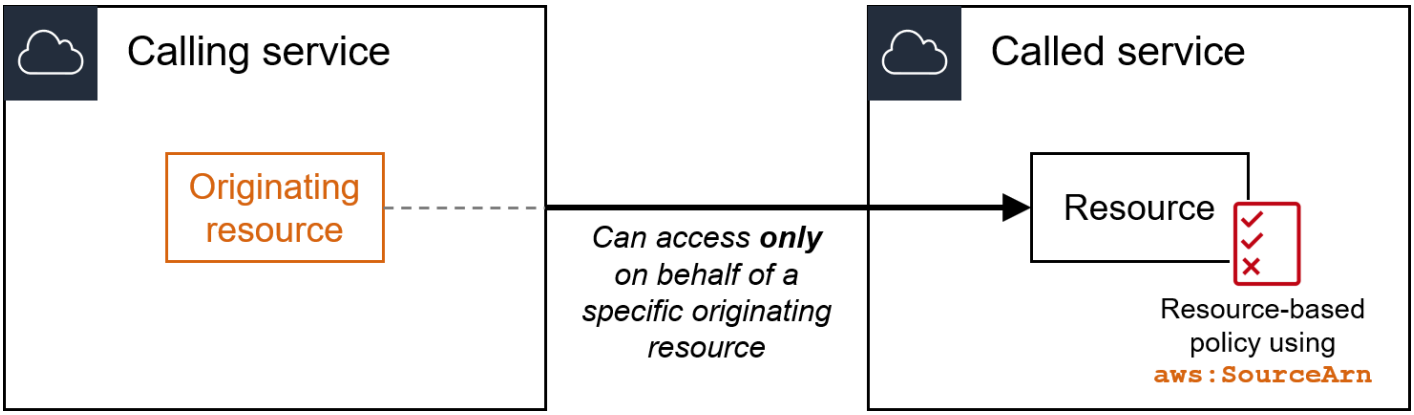
aws:SourceArn

Use this key to compare the Amazon Resource Name (ARN) of the resource making a service-to-service request with the ARN that you specify in the policy, but only when the request is made by an AWS service principal. When the source's ARN includes the account ID, it is not necessary to use `aws:SourceAccount` with `aws:SourceArn`.

This key does not work with the ARN of the principal making the request. Instead, use `aws:PrincipalArn`.

- **Availability** – This key is included in the request context only when the call to your resource is being made directly by an AWS service principal on behalf of a resource for which the configuration triggered the service-to-service request. The calling service passes the ARN of the original resource to the called service.

> **Note**
>
> This key provides a uniform mechanism for enforcing cross-service confused deputy control across AWS services. However, not all service integrations require the use of this global condition key. See the documentation of the AWS services you use for more information about service-specific mechanisms for mitigating cross-service confused deputy risks.



- **Data type** – ARN

  AWS recommends that you use ARN operators instead of string operators when comparing ARNs.
- **Value type** – Single-valued

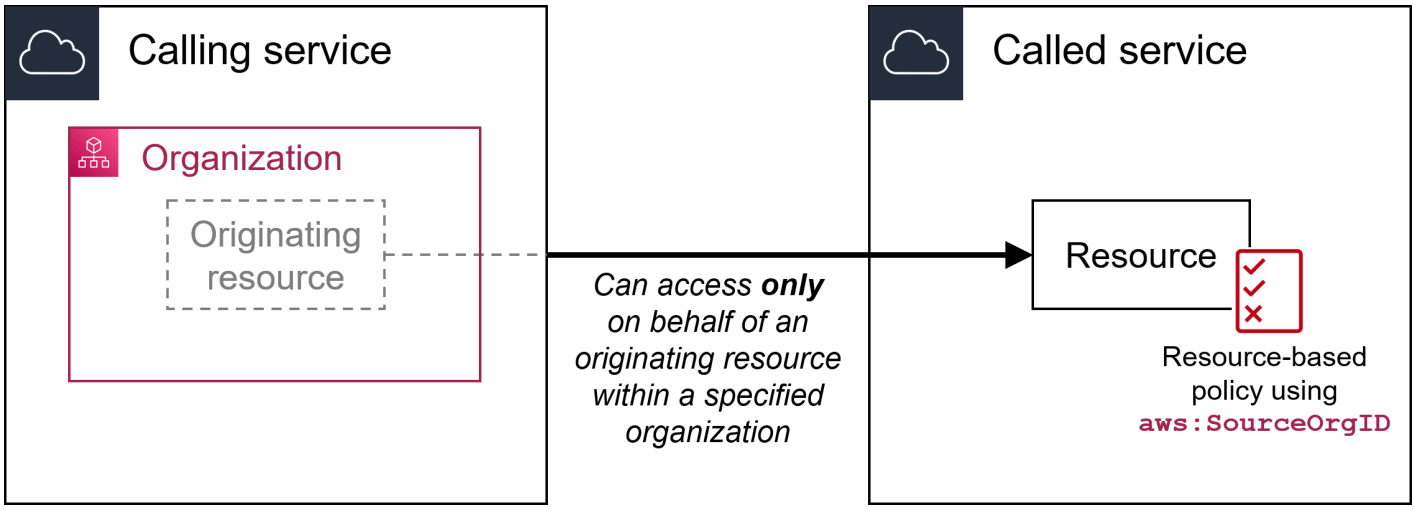
You can use this condition key to help make sure that a calling service can access your resource only when the request originates from a specific resource. When using a resource-based policy with an AWS service principal as the `Principal`, set this condition key's value to the ARN of the resource you want to restrict access to. For example, when an Amazon S3 bucket is configured to send notifications to an Amazon SNS topic, the Amazon S3 service invokes the `sns:Publish` API operation for all configured events. In the topic policy that allows the `sns:Publish` operation, set the value of the condition key to the ARN of the Amazon S3 bucket. For recommendations on when to use this condition key in resource-based policies, see the documentation for the AWS services you are using.

aws:SourceOrgID

Use this key to compare the organization ID of the resource making a service-to-service request with the organization ID that you specify in the policy, but only when the request is made by an AWS service principal. When you add and remove accounts to an organization in AWS Organizations, policies that include the `aws:SourceOrgID` key automatically include the correct accounts and you don't have to manually update the policies.

- **Availability** – This key is included in the request context only when the call to your resource is being made directly by an AWS service principal on behalf of a resource owned by an account which is a member of an organization. The calling service passes the organization ID of the original resource to the called service.

> **Note**
>
> This key provides a uniform mechanism for enforcing cross-service confused deputy control across AWS services. However, not all service integrations require the use of this global condition key. See the documentation of the AWS services you use for more information about service-specific mechanisms for mitigating cross-service confused deputy risks.



- **Data type** – String
- **Value type** – Single-valued

You can use this condition key to help make sure that a calling service can access your resource only when the request originates from a specific organization. For example, you can attach the following resource control policy (RCP) to deny requests by service principals against Amazon S3 buckets, unless they were triggered by a resource in the specified AWS organization. This policy applies the control only on requests by service principals (`"Bool": {"aws:PrincipalIsAWSService": "true"}`) that have the `aws:SourceAccount` key present (`"Null": {"aws:SourceAccount": "false"}`), so that service integrations that don't require the use of the key and calls by your principals aren't impacted. If the `aws:SourceAccount` key is present in the request context, the `Null` condition will evaluate to `true`, causing the `aws:SourceOrgID` key to be enforced. We use `aws:SourceAccount` instead of `aws:SourceOrgID` in the `Null` condition operator so that the control still applies if the request originates from an account that doesn't belong to an organization.

JSON

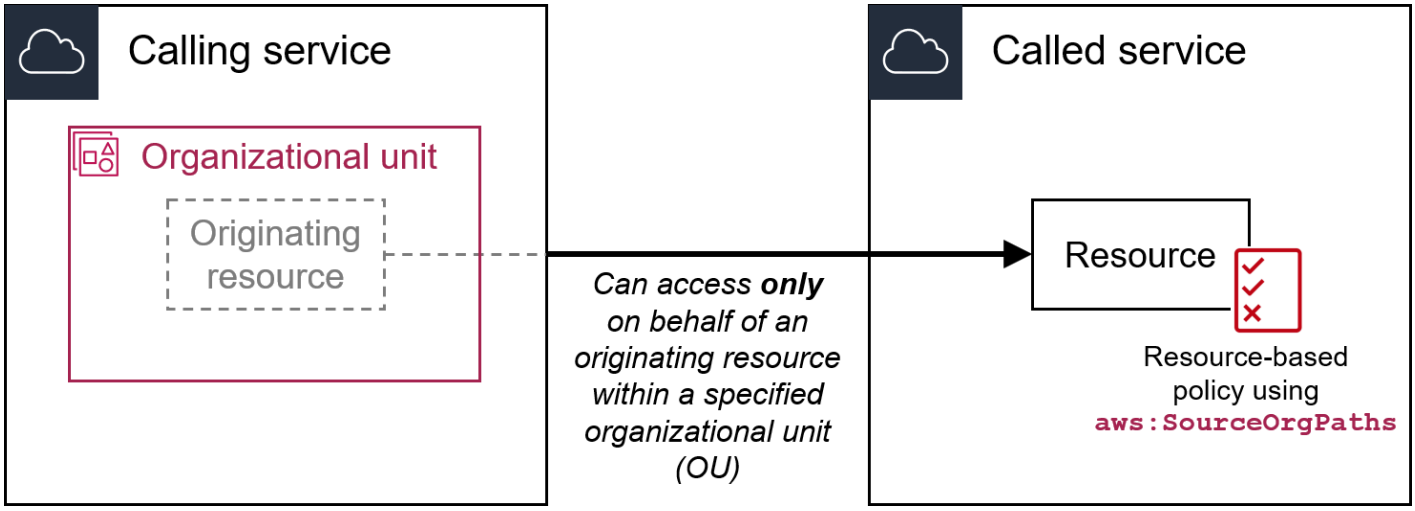
```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "RCPEnforceConfusedDeputyProtection",
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "s3:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceOrgID": "o-xxxxxxxxxx"
        },
        "Null": {
          "aws:SourceAccount": "false"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "true"
        }
      }
    }
  ]
}
```

aws:SourceOrgPaths

Use this key to compare the AWS Organizations path of the resource making a service-to-service request with the organizations path that you specify in the policy, but only when the request is made by an AWS service principal. An AWS Organizations path is a text representation of the structure of an AWS Organizations entity. For more information about using and understanding paths, see Understand the AWS Organizations entity path.

- **Availability** – This key is included in the request context only when the call to your resource is being made directly by an AWS service principal on behalf of a resource owned by an account which is a member of an organization. The calling service passes the organization path of the original resource to the called service.

> **Note**
>
> This key provides a uniform mechanism for enforcing cross-service confused deputy control across AWS services. However, not all service integrations require the use of this global condition key. See the documentation of the AWS services you use for more information about service-specific mechanisms for mitigating cross-service confused deputy risks.



- **Data type** – String (list)
- **Value type** – Multivalued

Use this condition key to help make sure that a calling service can access your resource only when the request originates from a specific organizational unit (OU) in AWS Organizations.

Similarly to `aws:SourceOrgID`, to help prevent impact on service integrations that don't require the use of this key, use the `Null` condition operator with the `aws:SourceAccount` condition key so that the control still applies if the request originates from an account that doesn't belong to an organization.

```
{
      "Condition": {
        "ForAllValues:StringNotLikeIfExists": {
            "aws:SourceOrgPaths": "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/ou-
ab12-22222222/"
        },
        "Null": {
          "aws:SourceAccount": "false"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "true"
        }
      }
}
```

`aws:SourceOrgPaths` is a multivalued condition key. Multivalued keys can have multiple values in the request context. You must use the `ForAnyValue` or `ForAllValues` set operators with string condition operators for this key. For more information about multivalued condition keys, see Set operators for multivalued context keys.

aws:UserAgent

Use this key to compare the requester's client application with the application that you specify in the policy.

- **Availability** – This key is always included in the request context.
- **Data type** – String
- **Value type** – Single-valued

Warning

This key should be used carefully. Since the `aws:UserAgent` value is provided by the caller in an HTTP header, unauthorized parties can use modified or custom browsers to provide any `aws:UserAgent` value that they choose. As a result, `aws:UserAgent` should not be used to prevent unauthorized parties from making direct AWS requests. You can use it to allow only specific client applications, and only after testing your policy.

aws:IsMcpServiceAction

Use this key to verify that the action being authorized is an MCP Service action. This key does not refer to actions taken by the MCP service to other AWS services.

- **Availability** – This key is included in the request context and set to True only when the MCP service is authorizing an MCP service action.
- **Data type** – Boolean
- **Value type** – Single-valued

Other cross-service condition keys

AWS STS supports SAML-based federation condition keys and cross-service condition keys for OIDC federation. These keys are available when a user who was federated using OIDC or SAML performs AWS operations in other services.

IAM and AWS STS condition context keys

You can use the `Condition` element in a JSON policy to test the value of keys that are included in the request context of all AWS requests. These keys provide information about the request itself or the resources that the request references. You can check that keys have specified values before allowing the action requested by the user. This gives you granular control over when your JSON policy statements match or don't match an incoming request. For information about how to use the `Condition` element in a JSON policy, see IAM JSON policy elements: Condition.

This topic describes the keys defined and provided by the IAM service (with an `iam:` prefix) and the AWS Security Token Service (AWS STS) service (with an `sts:` prefix). Several other AWS services also provide service-specific keys that are relevant to the actions and resources defined by that

service. For more information, see Actions, Resources, and Condition Keys for AWS Services. The documentation for a service that supports condition keys often has additional information. For example, for information about keys that you can use in policies for Amazon S3 resources, see Amazon S3 Policy Keys in the *Amazon Simple Storage Service User Guide*.

Topics

- Available keys for IAM
- Available keys for AWS OIDC federation
- Available keys for SAML-based AWS STS federation
- Cross-service SAML-based AWS STS federation context keys
- Available keys for AWS STS

Available keys for IAM

You can use the following condition keys in policies that control access to IAM resources:

iam:AccountPropertyNamespaces

Works with string operators.

Specifies the account property namespaces being read or modified in the request. Account properties are organized into namespaces (for example, `RoleManager`). When you call `PutAccountProperties` or `GetAccountProperties`, use this condition key to restrict which property namespaces a principal can access.

iam:AssociatedResourceArn

Works with ARN operators.

Specifies the ARN of the resource to which this role will be associated at the destination service. The resource usually belongs to the service to which the principal is passing the role. Sometimes, the resource might belong to a third service. For example, you might pass a role to Amazon EC2 Auto Scaling that they use on an Amazon EC2 instance. In this case, the condition would match the ARN of the Amazon EC2 instance.

This condition key applies to only the PassRole action in a policy. It can't be used to limit any other action.

> **Important**
>
> When using the `iam:AssociatedResourceArn` condition in a policy to restrict the PassRole action, special considerations apply if the policy is intended to define access for the AddRoleToInstanceProfile action. In this case, you cannot specify a Region or instance ID in the EC2 instance ARN. The ARN value must be `arn:aws:ec2:*:CallerAccountId:instance/*`. Using any other ARN value might lead to unexpected evaluation results.

Use this condition key in an identity-based policy to allow an entity to pass a role, but only if that role is associated with the specified resource. For example, you can allow an IAM user or role to pass any role to the Amazon EC2 service to be used with instances in the AWS account. The IAM user or role would not be allowed to pass roles to other services.

```
{
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "ec2.amazonaws.com"
        },
        "ArnLike": {
            "iam:AssociatedResourceARN": [
                "arn:aws:ec2:*:111122223333:instance/*"
            ]
        }
    }
}
```

> **Note**
>
> AWS services that support iam:PassedToService also support this condition key.

iam:AWSServiceName

Works with string operators.

Specifies the AWS service to which this role is attached.

This condition key is supported by the `CreateServiceLinkedRole` API operation.

> **Tip**
>
> For information about which services support using service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

In this example, you allow an entity to create a service-linked role using the `CreateServiceLinkedRole` API operation if the service name is *access-analyzer.amazonaws.com.*

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
       "Effect": "Allow",
       "Action": "iam:CreateServiceLinkedRole",
       "Resource": "*",
       "Condition": {
         "StringLike": {
           "iam:AWSServiceName": "access-analyzer.amazonaws.com"
         }
       }
     }]
 }
```

iam:FIDO-certification

Works with string operators.

Checks the MFA device FIDO certification level at the time of registration of a FIDO security key. The device certification is retrieved from the FIDO Alliance Metadata Service (MDS). If the certification status or level of your FIDO security key changes, it will not be updated unless the device is unregistered and registered again to fetch the updated certification information.

Possible values of L1, L1plus, L2, L2plus, L3, L3plus

In this example, you register a security key and retrieve the FIDO Level 1 plus certification for your device.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [{
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-certification": "L1plus"
                }
            }
        }
    ]

 }
```

iam:FIDO-FIPS-140-2-certification

Works with string operators.

Checks the MFA device FIPS-140-2 validation certification level at the time of registration of a FIDO security key. The device certification is retrieved from the FIDO Alliance Metadata Service (MDS). If the certification status or level of your FIDO security key changes, it will not be updated unless the device is unregistered and registered again to fetch the updated certification information.

Possible values of L1, L2, L3, L4

In this example, you register a security key and retrieve the FIPS-140-2 Level 2 certification for your device.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [{
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-2-certification": "L2"
                }
            }
        }
    ]

 }
```

iam:FIDO-FIPS-140-3-certification

Works with string operators.

Checks the MFA device FIPS-140-3 validation certification level at the time of registration of a FIDO security key. The device certification is retrieved from the FIDO Alliance Metadata Service (MDS). If the certification status or level of your FIDO security key changes, it will not be updated unless the device is unregistered and registered again to fetch the updated certification information.

Possible values of L1, L2, L3, L4

In this example, you register a security key and retrieve the FIPS-140-3 Level 3 certification for your device.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [{
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-3-certification": "L3"
                }
            }
        }
    ]

 }
```

iam:OrganizationsPolicyId

Works with string operators.

Checks that the policy with the specified AWS Organizations ID matches the policy used in the request. To view an example IAM policy that uses this condition key, see IAM: View service last accessed information for an AWS Organizations policy.

iam:PassedToService

Works with string operators.

Specifies the service principal of the service to which a role can be passed. This condition key applies to only the PassRole action in a policy. It can't be used to limit any other action.

When you use this condition key in a policy, specify the service using a service principal. A service principal is the name of a service that can be specified in the `Principal` element of a policy. This is the usual format: `SERVICE_NAME_URL.amazonaws.com`.

You can use `iam:PassedToService` to restrict your users so that they can pass roles only to specific services. For example, a user might create a service role that trusts CloudWatch to write log data to an Amazon S3 bucket on their behalf. Then the user must attach a permissions policy and a trust policy to the new service role. In this case, the trust policy must specify `cloudwatch.amazonaws.com` in the `Principal` element. To view a policy that allows the user to pass the role to CloudWatch, see IAM: Pass an IAM role to a specific AWS service.

By using this condition key, you can make sure that users create service roles only for the services that you specify. For example, if a user with the preceding policy attempts to create a service role for Amazon EC2, the operation will fail. The failure occurs because the user does not have permission to pass the role to Amazon EC2.

Sometimes you pass a role to a service that then passes the role to a different service. `iam:PassedToService` includes only the final service that assumes the role, not the intermediate service that passes the role.

> **Note**
>
> Some services do not support this condition key.

iam:PermissionsBoundary

Works with ARN operators.

Checks that the specified policy is attached as permissions boundary on the IAM principal resource. For more information, see Permissions boundaries for IAM entities

iam:PolicyARN

Works with ARN operators.

Checks the Amazon Resource Name (ARN) of a managed policy in requests that involve a managed policy. For more information, see Controlling access to policies.

iam:RoleTemplateARN

Works with ARN operators.

Specifies the ARN of the role template used to create or manage a role. When an AWS service acquires a role from a role template, the role template ARN is included as a context key in the authorization checks for `CreateRole`, `TagRole`, `PutRolePolicy`, `AttachRolePolicy`, `PutRolePermissionsBoundary`, and `GetRole`.

iam:RegisterSecurityKey

Works with string operators.

Checks the current state of MFA device enablement.

Possible values of `Create` or `Activate`.

In this example, you register a security key and retrieve the FIPS-140-3 Level 1 certification for your device.

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [{
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Create"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:EnableMFADevice",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:RegisterSecurityKey" : "Activate",
                    "iam:FIDO-FIPS-140-3-certification": "L1"
                }
            }
```

```
        }
    ]

}
```

iam:ResourceTag/*key-name*

Works with string operators.

Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.

> **Note**
>
> IAM and AWS STS support both the `iam:ResourceTag` IAM condition key and the `aws:ResourceTag` global condition key.

You can add custom attributes to IAM resources in the form of a key-value pair. For more information about tags for IAM resources, see the section called "Tags for IAM resources". You can use `ResourceTag` to control access to AWS resources, including IAM resources. However, because IAM does not support tags for groups, you cannot use tags to control access to groups.

This example shows how you might create an identity-based policy that allows deleting users with the **`status=terminated`** tag. To use this policy, replace the *`italicized placeholder text`* in the example policy with your own information. Then, follow the directions in create a policy or edit a policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "iam:DeleteUser",
        "Resource": "*",
        "Condition": {"StringEquals": {"iam:ResourceTag/status":
 "terminated"}}
    }]
}
```

iam:ServiceSpecificCredentialAgeDays

Works with numeric operators.

This condition key limits the creation of service-specific credentials based on their expiration settings. It allows you to control the maximum age, in days, of service-specific credentials that can be created.

The valid range for days is 1 to 36600 (minimum 1 day, maximum 36600 days).

This condition key is supported by the `CreateServiceSpecificCredential` API operation.

In this example, you allow a user to create service-specific credentials for the Amazon Bedrock service only if they expire within 90 days.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceSpecificCredential",
            "Resource": "arn:aws:iam::111122223333:user/username",
            "Condition": {
                "StringEquals": {
                    "iam:ServiceSpecificCredentialServiceName":
 "bedrock.amazonaws.com"
                },
                "NumericLessThanEquals": {
                    "iam:ServiceSpecificCredentialAgeDays": "90"
                }
            }
        }
    ]
}
```

iam:ServiceSpecificCredentialServiceName

Works with string operators.

Specifies which AWS services can be used when managing service-specific credentials. This condition key allows you to restrict which AWS services are permitted when managing service-specific credentials.

This condition key is supported by the following API operations:

- CreateServiceSpecificCredential
- DeleteServiceSpecificCredential
- ResetServiceSpecificCredential
- UpdateServiceSpecificCredential

The following services are supported for service-specific credentials with their exact value formatting:

- bedrock.amazonaws.com
- cassandra.amazonaws.com
- codecommit.amazonaws.com

In this example, you allow a user to create service-specific credentials using the CreateServiceSpecificCredential API operation only for the Amazon Bedrock service.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceSpecificCredential",
            "Resource": "arn:aws:iam::111122223333:user/username",
            "Condition": {
                "StringEquals": {
                    "iam:ServiceSpecificCredentialServiceName":
 "bedrock.amazonaws.com"
                }
            }
        }
    ]
}
```

iam:DelegationDuration

Works with numeric operators.

Filters access based on the duration of temporary access requested in a delegation request.

Product providers can use this condition key to control the maximum duration they allow in delegation requests sent to customers. The duration is specified in seconds and determines how long temporary credentials remain valid after the customer releases the exchange token. This helps product providers enforce internal policies around access duration limits based on their use case.

This condition key is supported by the `CreateDelegationRequest` API operation.

In this example, you allow creating delegation requests only if the requested duration is 7200 seconds (2 hours) or less.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateDelegationRequest",
            "Resource": "*",
            "Condition": {
                "NumericLessThanEquals": {
                    "iam:DelegationDuration": "7200"
                }
            }
        }
    ]
}
```

iam:NotificationChannel

Works with ARN operators.

Filters access based on the Amazon SNS topic ARN specified for receiving delegation request notifications.

Product providers can use this condition key to restrict which SNS topics can be used for delegation request notifications in the CreateDelegationRequest API call. Product providers must specify an SNS topic to receive state change notifications and exchange tokens. This ensures that notifications are sent only to approved channels within the product provider's organization.

This condition key is supported by the `CreateDelegationRequest` API operation.

In this example, you allow creating delegation requests only if they use a specific SNS topic for notifications.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateDelegationRequest",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "iam:NotificationChannel": "arn:aws:sns:us-
east-1:123456789012:delegation-notifications"
                }
            }
        }
    ]
}
```

iam:TemplateArn

Works with ARN operators.

Filters access based on the policy template ARN used to define permissions in a delegation request.

Product providers can use this condition key to control which policy templates can be used in the CreateDelegationRequest API call. Policy templates define the temporary permissions that product providers request in customer accounts. This allows product providers to restrict which registered policy templates can be used when creating delegation requests.

This condition key is supported by the `CreateDelegationRequest` API operation.

In this example, you allow creating delegation requests only if they use a policy template from a specific partner domain.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": "iam:CreateDelegationRequest",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "iam:TemplateArn": "arn:aws:iam::*:delegation-template/
partner_*"
                }
            }
        }
    ]
}
```

iam:DelegationRequestOwner

Works with ARN operators.

Filters access based on the AWS identity or principal that owns the delegation request.

Customers can use this condition key to control who can perform actions on delegation requests based on ownership. The owner of a delegation request is the AWS identity or principal in the customer account that initiated or received the delegation request.

This condition key is supported by the following API operations:

- `GetDelegationRequest`
- `AcceptDelegationRequest`
- `RejectDelegationRequest`
- `SendDelegationToken`
- `ListDelegationRequests`
- `UpdateDelegationRequest`

In this example, you allow users to manage only delegation requests that they own.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetDelegationRequest",
                "iam:AcceptDelegationRequest",
                "iam:RejectDelegationRequest",
                "iam:SendDelegationToken",
                "iam:UpdateDelegationRequest",
                "iam:ListDelegationRequests"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "iam:DelegationRequestOwner": "${aws:PrincipalArn}"
                }
            }
        }
    ]
}
```

Available keys for AWS OIDC federation

You can use OIDC federation to give temporary security credentials to users who have been authenticated through an OpenID Connect compatible identity provider (IdP) to an IAM OpenID Connect (OIDC) identity provider in your AWS account. Examples of such providers include GitHub, Amazon Cognito, Login with Amazon, and Google. Identity tokens and access tokens from your own IdP might be used, as well as service account tokens granted to Amazon Elastic Kubernetes Service workloads.

You can use AWS OIDC condition context keys to write policies that limit the access of federated principals to resources that are associated with a specific provider, app, or user. These keys are typically used in the trust policy for a role. Define condition keys using the name of the OIDC provider (`token.actions.githubusercontent.com`) followed by a claim (`:aud`): **`token.actions.githubusercontent.com:aud`**.

Some OIDC federation condition keys can be used in the role session to authorize resource access. If the value is **Yes** in the **Available in session** column, you can use these condition keys in policies to define what users are allowed to access in other AWS services. When a claim is not available in session, the OIDC condition context key can only be used in a role trust policy for the initial AssumeRoleWithWebIdentity authentication.

Select your IdP to see how claims from your IdP map to IAM condition context keys in AWS. More information on keys for GitHub and Google can be found under the **Default** tab.

Default

Default lists the standard OIDC claims and how they map to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column. Use this mapping if your IdP is not listed in the tab options.

GitHub Actions workflows and Google are some examples of IdPs that use the default implementation in their OIDC JWT ID token.

AWS STS condition key	IdP JWT claim	Available in session
amr	amr	Yes
aud	azp If no value is set for `azp`, the `aud` condition key maps to the `aud` claim.	Yes
email	email	No
oaud	aud	No
sub	sub	Yes

For more information about using OIDC condition context keys with GitHub, see Configuring a role for GitHub OIDC identity provider. For more information about Google `aud` and `azp` fields, see the Google Identity Platform OpenID Connect Guide.

amr

Works with string operators. The key is multivalued, meaning that you test it in a policy using condition set operators.

Example: `token.actions.githubusercontent.com:amr`

Authentication Methods Reference includes login information about the user. The key can contain the following values:

- If the user is unauthenticated, the key contains only `unauthenticated`.

- If the user is authenticated, the key contains the value `authenticated` and the name of the login provider used in the call (`accounts.google.com`).

aud

Works with string operators.

Examples:

- `accounts.google.com:aud`
- `token.actions.githubusercontent.com:aud`

Use the aud condition key to verify that the audience matches the one you specify in the policy. You can use the aud key with the sub key for the same identity provider.

This condition key is set from the following token fields:

- aud for OAuth 2.0 Google client IDs of your application, when the `azp` field is not set. When the `azp` field is set, the aud field matches the `accounts.google.com:oaud` condition key.
- `azp` when the `azp` field is set. This can happen for hybrid apps where a web application and Android app have a different OAuth 2.0 Google client ID but share the same Google APIs project.

When you write a policy using the `accounts.google.com:aud` condition key, you must know whether the app is a hybrid app that sets the `azp` field.

`azp` Field Not Set

The following example policy works for non-hybrid apps that do not set the `azp` field. In this case the Google ID Token aud field value matches both the `accounts.google.com:aud` and the `accounts.google.com:oaud` condition key values.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Federated": "accounts.google.com"},
            "Action": "sts:AssumeRoleWithWebIdentity",
```

```
            "Condition": {
                "StringEquals": {
                    "accounts.google.com:aud": "aud-value",
                    "accounts.google.com:oaud": "aud-value",
                    "accounts.google.com:sub": "sub-value"
                }
            }
        }
    ]
}
```

azp Field Set

The following example policy works for hybrid apps that do set the azp field. In this case, the Google ID Token aud field value matches only the `accounts.google.com:oaud` condition key value. The azp field value matches the `accounts.google.com:aud` condition key value.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Federated": "accounts.google.com"},
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "accounts.google.com:aud": "azp-value",
                    "accounts.google.com:oaud": "aud-value",
                    "accounts.google.com:sub": "sub-value"
                }
            }
        }
    ]
}
```

email

Works with string operators.

Example: `accounts.google.com:email`

This condition key validates the user's email address. The value of this claim might not be unique to this account and could change over time, therefore you should not use this value as the primary identifier to verify your user record.

oaud

Works with string operators.

Example: `accounts.google.com:oaud`

This key specifies the other audience (aud) that this ID token is intended for. It must be one of the OAuth 2.0 client IDs of your application.

sub

Works with string operators.

Examples:

- `accounts.google.com:sub`
- token.actions.githubusercontent.com:sub

Use these keys to verify that the subject matches the one that you specify in the policy. You can use the sub key with the aud key for the same identity provider.

In the following role trust policy, the sub condition key limits the role to the GitHub branch named demo.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/
token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
          "token.actions.githubusercontent.com:sub": "repo:org-name/repo-
name:ref:refs/heads/demo"
```

```
        }
      }
    }
  ]
}
```

AWS STS

This tab explains how AWS STS maps claims in JSON Web Tokens (JWTs) generated by GetWebIdentityToken to AWS STS condition context keys. You can use these keys in the destination role's trust policy to control access to the role. The **IdP JWT claim** column identifies the token claim mapped to each condition key.

For AWS STS, issuer-prefixed keys consist of your source account's issuer host, a colon, and the claim name. The issuer host is your issuer URL without the `https://` prefix. For example, the `sub` claim maps to `a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:sub`. For claims nested under the `https://sts.amazonaws.com/` namespace, omit the namespace—for example, use *`issuer-host`*`:org_id`. The `request_tags` and `source_identity` claims use the separate mappings shown in the table. To find your issuer URL, see Getting started with outbound identity federation.

The Available in session column indicates whether the same condition key is retained for policies evaluated on subsequent requests made with the role's temporary credentials.

For more information, see Federating AWS identities across AWS partitions.

AWS STS condition key	IdP JWT claim	Available in session
`issuer-host` `:sub`	`sub`	Yes
`issuer-host` `:aud`	`aud`	Yes
`issuer-host` `:jti`	`jti`	No
`issuer-host` `:aws_account`	`aws_account`	No
`issuer-host` `:source_region`	`source_region`	No. The value is retained as

AWS STS condition key	IdP JWT claim	Available in session
		aws:CrossPartitionSourceRegion.
issuer-host :principal_id	principal_id	No
issuer-host :original_session_exp	original_session_exp	No
issuer-host :invoked_by	invoked_by	No
issuer-host :mfa_present	mfa_present	No
issuer-host :mfa_age	mfa_age	No
issuer-host :org_id	org_id	No
issuer-host :ou_path	ou_path	No
issuer-host :identity_store_user_id	identity_store_user_id	No
issuer-host :identity_store_arn	identity_store_arn	No
issuer-host :ec2_source_instance_arn	ec2_source_instance_arn	No
issuer-host :ec2_role_delivery	ec2_role_delivery	No
issuer-host :ec2_instance_source_vpc	ec2_instance_source_vpc	No

AWS STS condition key	IdP JWT claim	Available in session
issuer-host :ec2_instance_source_private_ipv4	ec2_instance_source_private_ipv4	No
issuer-host :lambda_source_function_arn	lambda_source_function_arn	No
issuer-host :glue_credential_issuing_service	glue_credential_issuing_service	No
aws:RequestTag/ *tag-key*	A tag value in request_tags	No. The tag is retained as aws:PrincipalTag/*tag-key*.
aws:TagKeys	The tag keys in request_tags	No
sts:SourceIdentity	source_identity	No. The value is retained as aws:SourceIdentity.

Note

The iss, iat, exp, principal_tags, and federated_provider claims aren't mapped to condition keys for cross-partition role assumption.

issuer-host:sub

Data type: String. Works with string operators.

The ARN of the source IAM principal that requested the token. This claim is always present. For example, arn:aws:iam::123456789012:role/SourceWorkloadRole.

***issuer-host*:aud**

Data type: String. Works with string operators.

The audience for which the token was requested. This claim is always present. For example, `my-destination-app`.

***issuer-host*:jti**

Data type: String. Works with string operators.

The unique identifier of the token. This claim is always present. For example, `a1b2c3d4-5678-90ab-cdef-0123456789ab`.

***issuer-host*:aws_account**

Data type: String. Works with string operators.

The account ID of the source IAM principal. This claim is always present. For example, `123456789012`.

***issuer-host*:source_region**

Data type: String. Works with string operators.

The AWS Region where the token was requested. This claim is always present. For example, `us-east-1`.

***issuer-host*:principal_id**

Data type: ARN. Works with ARN operators.

The ARN of the source IAM principal that requested the token. This claim is always present. For example, `arn:aws:iam::123456789012:role/SourceWorkloadRole`.

***issuer-host*:original_session_exp**

Data type: Date. Works with date operators.

The expiration time of the source session credentials, in ISO 8601 format. This claim is present when the credentials have an expiration time. For example, `2026-10-02T02:03:26Z`.

***issuer-host*:invoked_by**

Data type: String. Works with string operators.

The AWS service acting on the customer's behalf by assuming the source role, or the last service in a forward access session. This claim is present only when the source session includes that service's identity. For example, `ec2.amazonaws.com`.

***issuer-host*:mfa_present**

Data type: String. Works with string operators.

Indicates that the source IAM principal authenticated with MFA. This claim is present only when MFA was used and has the string value `"true"`. It is never `"false"`.

To require MFA, use `StringEquals` with `"true"` in an `Allow` statement. To deny requests where the key is absent, use `Null` with `"true"` in a `Deny` statement.

***issuer-host*:mfa_age**

Data type: Numeric. Works with numeric operators.

The number of seconds since the source IAM principal authenticated with MFA, encoded as a string. This claim is present when MFA was used and the authentication age is available. For example, `"1800"`.

***issuer-host*:org_id**

Data type: String. Works with string operators.

The AWS Organizations ID of the source account. This claim is present when the account belongs to an organization and its ID is available. For example, `o-a1b2c3d4e5`.

***issuer-host*:ou_path**

Data type: String. Works with string operators and the `ForAnyValue` or `ForAllValues` set operator.

Value type: Multivalued.

The AWS Organizations paths for the source account. This claim is present when the source account's organization paths are available. For example, `["o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/"]`.

Use `ForAnyValue:StringLike` with a path ending in `/*`, such as `o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/*`, to match an organizational unit and its descendants. For more information, see aws:PrincipalOrgPaths.

***issuer-host*:identity_store_user_id**

Data type: String. Works with string operators.

The IAM Identity Center user ID associated with the source session. This claim is present when the session includes an IAM Identity Center user identity. For example, `94482488-3041-7026-1234-123456789012`. Use it with *issuer-host*`:identity_store_arn` to identify the user and identity store.

***issuer-host*:identity_store_arn**

Data type: ARN. Works with ARN operators.

The ARN of the IAM Identity Center identity store. This claim is present when the source session includes the identity store ARN. For example, `arn:aws:identitystore::123456789012:identitystore/d-1234567890`.

***issuer-host*:ec2_source_instance_arn**

Data type: ARN. Works with ARN operators.

The ARN of the Amazon EC2 instance that received the source role credentials. This claim is present when those credentials include the instance ARN. For example, `arn:aws:ec2:us-east-1:123456789012:instance/i-0123456789abcdef0`.

***issuer-host*:ec2_role_delivery**

Data type: Numeric. Works with numeric operators.

The instance metadata service (IMDS) version used to deliver the source role credentials. This claim is present when the credentials include the IMDS version: `1.0` for IMDSv1 or `2.0` for IMDSv2. For example, `2.0`.

***issuer-host*:ec2_instance_source_vpc**

Data type: String. Works with string operators.

The VPC ID of the Amazon EC2 instance that received the source role credentials. This claim is present when those credentials include the instance VPC ID. For example, `vpc-0123456789abcdef0`.

***issuer-host*:ec2_instance_source_private_ipv4**

Data type: IP address. Works with IP address operators.

The primary private IPv4 address of the Amazon EC2 instance that received the source role credentials. This claim is present when those credentials include the instance's private IPv4 address. For example, `10.0.0.15`.

***issuer-host*:lambda_source_function_arn**

Data type: ARN. Works with ARN operators.

The ARN of the Lambda function associated with the source credentials. This claim is present when those credentials include the function ARN. For example, `arn:aws:lambda:us-east-1:123456789012:function:MyFunction`.

***issuer-host*:glue_credential_issuing_service**

Data type: String. Works with string operators.

The AWS Glue service that issued the source credentials. This claim is present when credentials for a Glue job or developer endpoint include the service identifier. For example, `glue.amazonaws.com`.

aws:RequestTag/*tag-key*

Data type: String. Works with string operators.

A tag value from the token's `request_tags` claim. This key is present when the caller passes the corresponding tag to `GetWebIdentityToken`. For example, aws:RequestTag/project has the value `migration`.

The destination role's trust policy must allow `sts:TagSession`. Apply `aws:RequestTag/`*`tag-key`* conditions to that statement to restrict tag values.

aws:TagKeys

Data type: String. Works with string operators and the `ForAnyValue` or `ForAllValues` set operator.

Value type: Multivalued.

The tag keys in the token's `request_tags` claim. This key is present when the caller passes tags to `GetWebIdentityToken`. For example, `["project"]`.

The destination role's trust policy must allow `sts:TagSession`. Apply aws:TagKeys conditions to that statement to restrict tag keys.

sts:SourceIdentity

Data type: String. Works with string operators.

The source identity from the token's `source_identity` claim. This key is present when the source session has a source identity. For example, `alice`.

The destination role's trust policy must allow `sts:SetSourceIdentity`. Apply `sts:SourceIdentity` conditions to that statement to restrict source identity values.

The following example trust policy allows a single role in the source partition to assume the destination role when the token has the expected audience and was issued in `us-east-1`. The source partition is `aws` and the destination partition is `aws-cn`. The `Federated` principal identifies the OIDC provider in the destination account. Replace the issuer host in the provider ARN and condition keys with your source account's issuer host, and replace `your-audience` with the audience configured for your destination OIDC provider. Replace `us-east-1` with the source AWS Region you want to allow.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws-cn:iam::111122223333:oidc-
provider/a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:sub":
 "arn:aws:iam::123456789012:role/SourceWorkloadRole",
          "a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:aud":
 "your-audience",

 "a1d2b0fd-1177-4468-9351-2fEXAMPLE723.tokens.sts.global.api.aws:source_region":
 "us-east-1"
        }
      }
    }
  ]
```

```
}
```

> **Note**
>
> The `sub` condition matches every session of `SourceWorkloadRole`. To limit access to a specific Amazon EC2 instance or AWS Lambda function, also add a condition on *`issuer-host`*`:ec2_source_instance_arn` or *`issuer-host`*`:lambda_source_function_arn`. These claims are included only when the source workload runs on that service. To trust any principal from a source organization instead of a single role, use *`issuer-host`*`:org_id`.

Amazon Cognito

This tab explains how Amazon Cognito maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

For roles used by Amazon Cognito, keys are defined using `cognito-identity.amazonaws.com` followed by the claim.

For more information about identity pool claim mapping, see Default provider mappings in the *Amazon Cognito Developer Guide*. For more information about user pool claim mapping see Using the ID token in the *Amazon Cognito Developer Guide*.

AWS STS condition key	IdP JWT claim	Available in session
amr	amr	Yes
aud	aud	Yes
oaud	aud	No
sub	sub	Yes

amr

Works with string operators. The key is multivalued, meaning that you test it in a policy using condition set operators.

Example – `cognito-identity.amazonaws.com:amr`

Authentication Methods Reference includes login information about the user. The key can contain the following values:

- If the user is unauthenticated, the key contains only `unauthenticated`.
- If the user is authenticated, the key contains the value `authenticated` and the name of the login provider used in the call (`cognito-identity.amazonaws.com`).

As an example, the following condition in the trust policy for an Amazon Cognito role tests whether the user is unauthenticated.

```
"Condition": {
  "StringEquals":
    { "cognito-identity.amazonaws.com:aud": "us-east-2:identity-pool-id" },
  "ForAnyValue:StringLike":
    { "cognito-identity.amazonaws.com:amr": "unauthenticated" }
}
```

aud

Works with string operators.

Example – `cognito-identity.amazonaws.com:aud`

The user pool app client that authenticated your user. Amazon Cognito renders the same value in the access token `client_id` claim.

oaud

Works with string operators.

Example – `cognito-identity.amazonaws.com:oaud`

The user pool app client that authenticated your user. Amazon Cognito renders the same value in the access token `client_id` claim.

sub

Works with string operators.

Example – `cognito-identity.amazonaws.com:sub`

A unique identifier (UUID), or subject, for the authenticated user. The username might not be unique in your user pool. The sub claim is the best way to identify a given user. You can use the sub key with the aud key for the same identity provider.

```
"Condition": {
         "StringEquals": {
            "cognito-identity.amazonaws.com:aud": "us-east-1:12345678-abcd-abcd-
abcd-123456790ab",
            "cognito-identity.amazonaws.com:sub": [
               "us-east-1:12345678-1234-1234-1234-123456790ab",
               "us-east-1:98765432-1234-1234-1243-123456790ab"
            ]
         }
      }
```

Login with Amazon

This tab explains how Login with Amazon maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

AWS STS condition key	IdP JWT claim	Available in session
app_id	Application ID	Yes
sub	User ID	Yes
user_id	User ID	Yes

app_id

Works with string operators.

Example – `www.amazon.com:app_id`

This key specifies the audience context that matches the aud field used by other identity providers.

sub

Works with string operators.

Example – `www.amazon.com:sub`

This key verifies that the user ID matches the one that you specify in the policy. You can use the `sub` key with the `aud` key for the same identity provider.

user_id

Works with string operators.

Example – `www.amazon.com:user_id`

This key specifies the audience context that matches the `aud` field used by other identity providers.You can use the `user_id` key with the `id` key for the same identity provider.

Facebook

This tab explains how Facebook maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

AWS STS condition key	IdP JWT claim	Available in session
app_id	Application ID	Yes
id	id	Yes

app_id

Works with string operators.

Example – `graph.facebook.com:app_id`

This key verifies that the audience context matches the `aud` field used by other identity providers.

id

Works with string operators.

Example – `graph.facebook.com:id`

This key verified that the application (or site) ID matches the one that you specify in the policy.

GitHub

This tab explains how GitHub Actions maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

AWS STS condition key	IdP JWT claim	Available in session
actor	actor	No
actor_id	actor_id	No
job_workflow_ref	job_workflow_ref	No
repository	repository	No
repository_id	repository_id	No
repository_owner_id	repository_owner_id	No
workflow	workflow	No
ref	ref	No
environment	environment	No
enterprise_id	enterprise_id	No

actor

Works with string operators.

Example – `token.actions.githubusercontent.com:actor`

This key identifies the personal account that initiated the workflow run. Use this to restrict access to specific actors.

actor_id

Works with string operators.

Example – `token.actions.githubusercontent.com:actor_id`

This key verifies the ID of the personal account that initiated the workflow run. Actor IDs are generated by GitHub and are immutable.

job_workflow_ref

Works with string operators.

Example – `token.actions.githubusercontent.com:job_workflow_ref`

This key contains the reference path to the reusable workflow for jobs using a reusable workflow. Use this to restrict access to specific workflows and to make sure only approved workflows can assume roles.

repository

Works with string operators.

Example – `token.actions.githubusercontent.com:repository`

This key identifies the repository from where the workflow is running. Use this to limit access to specific GitHub repositories.

repository_id

Works with string operators.

Example – `token.actions.githubusercontent.com:repository_id`

This key verifies the ID of the repository from where the workflow is running. Repository IDs are immutable and don't change even if the repository is renamed.

repository_owner_id

Works with string operators.

Example – `token.actions.githubusercontent.com:repository_owner_id`

This key verifies the ID of the organization in which the repository is stored. The repository owner ID is a stable, unique identifier that does not change.

workflow

Works with string operators.

Example – `token.actions.githubusercontent.com:workflow`

This key contains the name of the workflow. Use this to restrict access to specific workflows within your repositories.

ref

Works with string operators.

Example – `token.actions.githubusercontent.com:ref`

This key identifies the git ref (branch or tag) that triggered the workflow run. Use this to limit access based on specific branches, such as allowing only `main` or `production` branches.

environment

Works with string operators.

Example – `token.actions.githubusercontent.com:environment`

This key contains the name of the environment used by the job. Use this to implement environment-based access controls, such as separate permissions for development, staging, and production environments.

> **Note**
>
> If the environment claim is included in your trust policy, an environment must be configured and provided in the GitHub workflow.

enterprise_id

Works with string operators.

Example – `token.actions.githubusercontent.com:enterprise_id`

This key verifies the ID of the enterprise that contains the repository from where the workflow is running. Use this to make sure access is limited to repositories within your GitHub Enterprise organization.

> **Use immutable identifiers, not names**
>
> On GitHub, repository, organization, and user names can change. A name that is freed by renaming or deletion can be claimed by a different account. Policies that rely solely

on mutable name-based claims (such as `repository` or `actor`) could grant access to unintended identities. To help protect against this, use the immutable identifiers that GitHub provides, such as `repository_id`, `repository_owner_id`, or `actor_id`, in your role trust policies or resource control policies.

The following example trust policy uses custom claims in GitHub OIDC token to limit access to a role.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Federated": "arn:aws:iam::111122223333:oidc-provider/
token.actions.githubusercontent.com"
         },
         "Action": "sts:AssumeRoleWithWebIdentity",
         "Condition": {
            "StringEquals": {
                "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
                "token.actions.githubusercontent.com:repository_owner_id": "123456",
                "token.actions.githubusercontent.com:repository_id": "1296269",
                "token.actions.githubusercontent.com:actor_id": "1",
                "token.actions.githubusercontent.com:ref": "refs/heads/main",
                "token.actions.githubusercontent.com:enterprise_id": "345"
               }
          }
        }
    ]
}
```

Google

This tab explains how Google maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

AWS STS condition key	IdP JWT claim	Available in session
google/organizatio n_number	google:organization_number	No

google/organization_number

Works with numeric operators.

Example – `accounts.google.com:google/organization_number`

This key verifies that a token represents a Google identity belonging to a specific Google Cloud or Google Workspace organization. Use this to restrict access to users from specific organizations, ensuring that only identities from your organization can assume the role.

The following example trust policy uses the `google/organization_number` claim to limit access to a role.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Federated": "accounts.google.com"
         },
         "Action": "sts:AssumeRoleWithWebIdentity",
         "Condition": {
            "NumericEquals": {
                "accounts.google.com:google/organization_number": "123456"
               }
           }
        }
    ]
}
```

CircleCI

This tab explains how CircleCI maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

AWS STS condition key	IdP JWT claim	Available in session
oidc.circleci.com/project-id	oidc.circleci.com/project-id	No

oidc.circleci.com/project-id

Works with string operators.

Example – `circleci-issuer-url:oidc.circleci.com/project-id`

This key identifies the CircleCI project in which the job is running. Its value is a string containing a UUID that uniquely identifies the CircleCI project. Use this to restrict access to specific CircleCI projects.

The following example trust policy uses the `oidc.circleci.com/project-id` claim to limit access to a role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/oidc.circleci.com/
org/12345"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.circleci.com/org/12345:aud": "sts.amazonaws.com",
          "oidc.circleci.com/org/12345:oidc.circleci.com/project-id": "76543210-
ba98-fedc-3210-edcba0987654"
        }
      }
```

```
    }
  ]
}
```

Oracle Cloud Infrastructure (OCI)

This tab explains how Oracle Cloud Infrastructure maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

AWS STS condition key	IdP JWT claim	Available in session
rpst_id	rpst_id	No

rpst_id

Works with string operators.

Example – `oci-issuer-url:rpst_id`

This key uniquely identifies the resource principal in OCI. Use this to restrict access to specific OCI resource principals. The rpst_id (Resource Principal Session Token ID) provides a stable identifier for OCI resource-based authentication.

The following example trust policy uses the `rpst_id` claim to limit access to a role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/idcs-
abc123ef5678901234abcd.identity.oraclecloud.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "idcs-abc123ef5678901234abcd.identity.oraclecloud.com:aud":
 "sts.amazonaws.com",
```

```
          "idcs-abc123ef5678901234abcd.identity.oraclecloud.com:rpst_id": "your-
rpst-id"
        }
      }
    }
  ]
}
```

GitLab.com

This tab explains how GitLab.com CI/CD maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

> **Note**
>
> These custom condition keys are only supported when using the GitLab.com multi-tenant SaaS instance as an OIDC identity provider. You cannot use these condition keys with GitLab Dedicated or GitLab Self-Managed instances.

AWS STS condition key	IdP JWT claim	Available in session
namespace_id	namespace_id	No
project_id	project_id	No
user_id	user_id	No
user_login	user_login	No
user_email	user_email	No
user_access_level	user_access_level	No
ref_protected	ref_protected	No
pipeline_source	pipeline_source	No
runner_environment	runner_environment	No

namespace_id

Works with string operators.

Example – `gitlab.com:namespace_id`

This key identifies the namespace (group) ID of the project running the CI/CD job. The namespace ID is a stable, unique identifier that does not change if the group is renamed.

project_id

Works with string operators.

Example – `gitlab.com:project_id`

This key identifies the ID of the project running the CI/CD job. The project ID is a stable, unique identifier that does not change if the project is renamed or moved.

user_id

Works with string operators.

Example – `gitlab.com:user_id`

This key identifies the ID of the user executing the CI/CD job. The user ID is a stable, unique identifier.

user_login

Works with string operators.

Example – `gitlab.com:user_login`

This key identifies the username of the user executing the CI/CD job. Usernames can be changed; for a stable identifier, use `user_id` instead.

user_email

Works with string operators.

Example – `gitlab.com:user_email`

This key identifies the email address of the user executing the CI/CD job. The value of this claim might change over time; for a stable identifier, use `user_id` instead.

user_access_level

Works with string operators.

Example – `gitlab.com:user_access_level`

This key identifies the access level of the user within the project (for example, `maintainer`, `developer`, `owner`).

ref_protected

Works with string operators.

Example – `gitlab.com:ref_protected`

This key indicates whether the Git ref that triggered the job is protected. The value is `true` if protected, `false` otherwise.

pipeline_source

Works with string operators.

Example – `gitlab.com:pipeline_source`

This key identifies the source that triggered the pipeline (for example, `push`, `web`, `schedule`, `api`, `merge_request_event`).

runner_environment

Works with string operators.

Example – `gitlab.com:runner_environment`

This key identifies the type of runner used by the CI/CD job.

Important

On GitLab.com SaaS, deleted group or project paths could previously be recreated by different users, which could cause policies relying solely on path-based claims (such as `sub`) to grant access to unintended identities. As of June 1, 2026, GitLab has deployed a platform-side update that prevents new CI ID tokens from being issued for project paths that previously issued tokens, reducing this risk going forward. As an additional

layer of protection, AWS and GitLab recommend including conditions on stable, unique identifiers — such as `namespace_id` or `project_id` — in your role trust policies or resource control policies.

The following example trust policy uses the `namespace_id` claim together with `sub` to limit access to a specific GitLab.com project and branch.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/gitlab.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "gitlab.com:sub": "project_path:mygroup/
myproject:ref_type:branch:ref:main",
          "gitlab.com:namespace_id": "12345"
        }
      }
    }
  ]
}
```

Buildkite

This tab explains how Buildkite maps OIDC claims to AWS STS condition context keys in AWS. You can use these keys to control access to a role. To do that, compare the **AWS STS condition keys** to the values in the **IdP JWT claim** column.

AWS STS condition key	IdP JWT claim	Available in session
organization_slug	organization_slug	No
organization_id	organization_id	No
pipeline_slug	pipeline_slug	No

AWS STS condition key	IdP JWT claim	Available in session
pipeline_id	pipeline_id	No
cluster_name	cluster_name	No
cluster_id	cluster_id	No
build_branch	build_branch	No

organization_slug

Works with string operators.

Example – `agent.buildkite.com:organization_slug`

This key identifies the Buildkite organization from which the job is running. Use this to restrict access to specific organizations.

organization_id

Works with string operators.

Example – `agent.buildkite.com:organization_id`

This key verifies the ID of the Buildkite organization from which the job is running. Organization IDs are generated by Buildkite and are immutable.

pipeline_slug

Works with string operators.

Example – `agent.buildkite.com:pipeline_slug`

This key identifies the pipeline in which the job is running. Use this to restrict access to specific Buildkite pipelines.

pipeline_id

Works with string operators.

Example – `agent.buildkite.com:pipeline_id`

This key verifies the ID of the pipeline in which the job is running. Pipeline IDs are generated by Buildkite and are immutable.

cluster_name

Works with string operators.

Example – `agent.buildkite.com:cluster_name`

This key identifies the cluster on which the job is running. Use this to restrict access to specific Buildkite clusters.

cluster_id

Works with string operators.

Example – `agent.buildkite.com:cluster_id`

This key verifies the ID of the cluster on which the job is running. Cluster IDs are generated by Buildkite and are immutable.

build_branch

Works with string operators.

Example – `agent.buildkite.com:build_branch`

This key identifies the git branch that triggered the build. Use this to limit access based on specific branches, such as allowing only `main` or `production` branches.

⚠ Use immutable identifiers, not slugs

On Buildkite, a deleted organization or pipeline slug can be immediately re-registered by a different organization or pipeline. Buildkite issues OIDC tokens for these slugs from the same issuer (`agent.buildkite.com`). As a result, policies that rely solely on slug-based claims (such as `organization_slug` or `pipeline_slug`) could grant access to unintended identities. To help protect against this, use stable, immutable identifiers, such as `organization_id` or `pipeline_id`, in your role trust policies or resource control policies.

The following example trust policy uses custom claims in the Buildkite OIDC token to limit access to a role.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Federated": "arn:aws:iam::123456789012:oidc-provider/
agent.buildkite.com"
        },
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {
            "StringEquals": {
                "agent.buildkite.com:aud": "sts.amazonaws.com",
                "agent.buildkite.com:organization_id": "0191e7a0-1234-7abc-
def0-123456789abc",
                "agent.buildkite.com:pipeline_id": "0191e7a1-5678-7abc-
def0-123456789abc",
                "agent.buildkite.com:build_branch": "main"
            }
        }
    }
    ]
}
```

More information about OIDC federation

- Amazon Cognito User Guide
- OIDC federation

Available keys for SAML-based AWS STS federation

If you are working with SAML-based federation using AWS Security Token Service (AWS STS), you can include additional condition keys in the policy.

SAML role trust policies

In the trust policy of a role, you can include the following keys, which help you establish whether the caller is allowed to assume the role. Except for `saml:doc`, all the values are derived from the SAML assertion. All items in the list are available in the IAM console visual editor when you create

or edit a policy with conditions. Items marked with `[]` *can* have a value that is a list of the specified type.

saml:aud

Works with string operators.

An endpoint URL to which SAML assertions are presented. The value for this key comes from the `SAML Recipient` field in the assertion, *not* the `Audience` field.

saml:commonName[]

Works with string operators.

This is a `commonName` attribute.

saml:cn[]

Works with string operators.

This is an `eduOrg` attribute.

saml:doc

Works with string operators.

This represents the principal that was used to assume the role. The format is *`account-ID`*`/`*`provider-friendly-name`*, such as `123456789012/SAMLProviderName`. The *account-ID* value refers to the account that owns the SAML provider.

saml:edupersonaffiliation[]

Works with string operators.

This is an `eduPerson` attribute.

saml:edupersonassurance[]

Works with string operators.

This is an `eduPerson` attribute.

saml:edupersonentitlement[]

Works with string operators.

This is an `eduPerson` attribute.

saml:edupersonnickname[]

Works with string operators.

This is an eduPerson attribute.

saml:edupersonorgdn

Works with string operators.

This is an eduPerson attribute.

saml:edupersonorgunitdn[]

Works with string operators.

This is an eduPerson attribute.

saml:edupersonprimaryaffiliation

Works with string operators.

This is an eduPerson attribute.

saml:edupersonprimaryorgunitdn

Works with string operators.

This is an eduPerson attribute.

saml:edupersonprincipalname

Works with string operators.

This is an eduPerson attribute.

saml:edupersonscopedaffiliation[]

Works with string operators.

This is an eduPerson attribute.

saml:edupersontargetedid[]

Works with string operators.

This is an eduPerson attribute.

saml:eduorghomepageuri[]

Works with string operators.

This is an `eduOrg` attribute.

saml:eduorgidentityauthnpolicyuri[]

Works with string operators.

This is an `eduOrg` attribute.

saml:eduorglegalname[]

Works with string operators.

This is an `eduOrg` attribute.

saml:eduorgsuperioruri[]

Works with string operators.

This is an `eduOrg` attribute.

saml:eduorgwhitepagesuri[]

Works with string operators.

This is an `eduOrg` attribute.

saml:givenName[]

Works with string operators.

This is a `givenName` attribute.

saml:iss

Works with string operators.

The issuer, which is represented by a URN.

saml:mail[]

Works with string operators.

This is a `mail` attribute.

saml:name[]

Works with string operators.

This is a `name` attribute.

saml:namequalifier

Works with string operators.

A hash value based on the friendly name of the SAML provider. The value is the concatenation of the following values, in order and separated by a '/' character:

1. The `Issuer` response value (`saml:iss`)
2. The `AWS` account ID
3. The friendly name (the last part of the ARN) of the SAML provider in IAM

The concatenation of the account ID and friendly name of the SAML provider is available to IAM policies as the key `saml:doc`. For more information, see Uniquely identifying users in SAML-based federation.

saml:organizationStatus[]

Works with string operators.

This is an `organizationStatus` attribute.

saml:primaryGroupSID[]

Works with string operators.

This is a `primaryGroupSID` attribute.

saml:sub

Works with string operators.

This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).

saml:sub_type

Works with string operators.

This key can have the value `persistent`, `transient`, or consist of the full `Format` URI from the `Subject` and `NameID` elements used in your SAML assertion. A value of `persistent`

indicates that the value in `saml:sub` is the same for a user between sessions. If the value is `transient`, the user has a different `saml:sub` value for each session. For information about the `NameID` element's `Format` attribute, see Configure SAML assertions for the authentication response.

saml:surname[]

Works with string operators.

This is a `surnameuid` attribute.

saml:uid[]

Works with string operators.

This is a `uid` attribute.

saml:x500UniqueIdentifier[]

Works with string operators.

This is an `x500UniqueIdentifier` attribute.

For general information about `eduPerson` and `eduOrg` attributes, see the REFEDS Wiki website. For a list of `eduPerson` attributes, see eduPerson Object Class Specification (201602).

Condition keys whose type is a list can include multiple values. To create conditions in the policy for list values, you can use set operators (`ForAllValues`, `ForAnyValue`). For example, to allow any user whose affiliation is "faculty" or "staff" (but not "student"), you might use a condition like the following:

```
"Condition": {
   "ForAllValues:StringLike": {
     "saml:edupersonaffiliation":[ "faculty", "staff"]
   }
}
```

Cross-service SAML-based AWS STS federation context keys

Some SAML-based federation condition keys can be used in subsequent requests to authorize AWS operations in other services and `AssumeRole` calls. These are the following condition keys that can be used in role trust policies when federated principals assume another role, and in resource

policies from other AWS services to authorize resource access by federated principals. For more information about using these keys, see About SAML 2.0-based federation.

Select a condition key to see the description.

- saml:namequalifier
- saml:sub
- saml:sub_type

> **Note**
>
> No other SAML-based federation condition keys are available for use after the initial external identity provider (IdP) authentication response.

Available keys for AWS STS

You can use the following condition keys in IAM role trust policies for roles that are assumed using AWS Security Token Service (AWS STS) operations.

saml:sub

Works with string operators.

This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization (for example, `_cbb88bf52c2510eabe00c1642d4643f41430fe25e3`).

sts:AWSServiceName

Works with string operators.

Use this key to specify a service where a bearer token can be used. When you use this condition key in a policy, specify the service using a service principal. A service principal is the name of a service that can be specified in the `Principal` element of a policy. For example, `codeartifact.amazonaws.com` is the AWS CodeArtifact service principal.

Availability – This key is present in requests that get a bearer token. You cannot make a direct call to AWS STS to get a bearer token. When you perform some operations in other services, the service requests the bearer token on your behalf.

Some AWS services require that you have permission to get an AWS STS service bearer token before you can access their resources programmatically. For example, AWS CodeArtifact requires principals to use bearer tokens to perform some operations. The `aws codeartifact get-authorization-token` command returns a bearer token. You can then use the bearer token to perform AWS CodeArtifact operations. For more information about bearer tokens, see Service bearer tokens.

You can use this condition key to allow principals to get a bearer token for use with a specific service.

sts:DurationSeconds

Works with numeric operators.

Use this key to specify the duration (in seconds) that a principal can use when getting an AWS AWS STS bearer token or a JSON Web Token from the GetWebIdentityToken API.

Availability – This key is present in requests that get a bearer token or a JSON Web Token from the GetWebIdentityToken API. You cannot make a direct call to AWS STS to get a bearer token. When you perform some operations in other services, the service requests the bearer token on your behalf. The key is not applicable for AWS STS assume-role operations.

Some AWS services require that you have permission to get an AWS STS service bearer token before you can access their resources programmatically. For example, AWS CodeArtifact requires principals to use bearer tokens to perform some operations. The `aws codeartifact get-authorization-token` command returns a bearer token. You can then use the bearer token to perform AWS CodeArtifact operations. For more information about bearer tokens, see Service bearer tokens.

sts:IdentityTokenAudience

Works with string operators.

Use this key to specify the audiences for which an IAM principal can request JSON Web Tokens (JWTs) using the GetWebIdentityToken API. External services validate the audience ("aud") claim in the JSON Web Token to make sure the token was intended for them.

Availability – This key is present in requests to the GetWebIdentityToken API which is used to obtain JSON Web Tokens (JWTs) for authentication with external services.

When you use this condition key in a policy, specify the audience value that matches the intended recipient's identifier (for example, https://api.example.com).

This key is multivalued. In an `Allow` statement, use `ForAllValues:StringEquals` to require every requested audience to match an allowed value. Include a `Null` condition with `false` to require the key to be present and nonempty. For more information, see Set operators for multivalued context keys.

The following example policy allows a principal to request tokens for the specified external services:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:GetWebIdentityToken",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "sts:IdentityTokenAudience": [
                        "https://api2.example.com",
                        "https://api1.example.com"
                    ]
                },
                "Null": {
                    "sts:IdentityTokenAudience": "false"
                }
            }
        }
    ]
}
```

sts:SigningAlgorithm

Works with string operators.

Use this key to specify the cryptographic algorithm AWS AWS STS uses to sign JSON Web Tokens (JWTs) generated by the GetWebIdentityToken API. When you use this condition key in a policy, specify either ES384 (ECDSA with P-384 curve and SHA-384) or RS256 (RSA with SHA-256).

Availability – This key is present in requests to the GetWebIdentityToken API which is used to obtain JSON Web Tokens (JWTs) for authentication with external services.

You can use this condition key to enforce that IAM principals request tokens using signing algorithms compatible with your security requirements or the external services you integrate with. ES384 provides optimal security and performance, while RS256 offers broader compatibility with systems that do not support ECDSA.

The following example policy requires principals to use the ES384 signing algorithm:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:GetWebIdentityToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:SigningAlgorithm": "ES384"
                }
            }
        }
    ]
}
```

sts:ExternalId

Works with string operators.

Use this key to require that a principal provide a specific identifier when assuming an IAM role.

Availability – This key is present in the request when the principal provides an external ID while assuming a role using the AWS CLI or AWS API.

A unique identifier that might be required when you assume a role in another account. If the administrator of the account to which the role belongs provided you with an external ID, then provide that value in the `ExternalId` parameter. This value can be any string, such as a passphrase or account number. The primary function of the external ID is to address and prevent the confused deputy problem. For more information about the external ID and the confused deputy problem, see Access to AWS accounts owned by third parties.

The `ExternalId` value must have a minimum of 2 characters and a maximum of 1,224 characters. The value must be alphanumeric without white space. It can also include the

following symbols: plus (+), equal (=), comma (,), period (.), at (@), colon (:), forward slash (/), and hyphen (-).

sts:RequestContext/*context-key*

Works with string operators.

Use this key to compare the session context key-value pairs that are embedded in the trusted token issuer signed context assertion passed in the request with the context key-values specified in the role trust policy.

Availability – This key is present in the request when a context assertion is provided in the `ProvidedContexts` request parameter while assuming a role using the AWS STS AssumeRole API operation.

This context key is formatted as `"sts:RequestContext/context-key":"context-value"` where `context-key` and `context-value` are a context key-value pair. When multiple context keys are embedded in the signed context assertion passed in the request, there is one context key for each key-value pair. You must grant permission for the `sts:SetContext` action in the role trust policy to allow a principal to set context keys within the resulting session token. To learn more about the supported IAM Identity Center context keys that can be used with this key, see AWS STS condition keys for IAM Identity Center in the *AWS IAM Identity Center User Guide*.

You can use this key in a role trust policy to enforce fine-grained access control based on the user or their attributes when they assume a role. After the role is assumed, activity appears in the AWS CloudTrail logs within the `AdditionalEventData` attribute, containing the session context key-value pairs that were set by the context provider in the assume role request. This makes it easier for administrators to differentiate between role sessions when a role is used by different principals. The key-value pairs are set by the specified context provider, not by AWS CloudTrail or AWS STS. This gives the context provider control over what context is included in the CloudTrail logs and session information.

sts:RequestContextProviders

Works with ARN operators.

Use this key to compare the context provider ARN in the request with the context provider ARN specified in the role trust policy.

Availability – This key is present in the request when a context assertion is provided in the `ProvidedContexts` request parameter while assuming a role using the AWS STS AssumeRole API operation.

The following example condition checks that the context provider ARN passed in the request matches the ARN specified in the role trust policy condition. We recommend you add a null check with `ForAllValues` to prevent missing context keys or context keys with empty values from evaluating to true. For details, see Condition operator to check existence of condition keys.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Action": "sts:SetContext",
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "ForAllValues:ArnEquals": {
        "sts:RequestContextProviders": [
          "arn:aws:iam::aws:contextProvider/IdentityCenter"
        ]
      },
      "Null": {
        "sts:RequestContextProviders": "false"
      }
    }
  }
}
```

sts:RoleAuthorizedByIdp

Works with Boolean operators.

Use this key to verify that the identity provider (IdP) explicitly authorized the requested role through the `https://aws.amazon.com/roles` claim in the OIDC token. The roles claim contains the IAM role ARNs that the token is allowed to assume. AWS STS supports two formats for the claim value:

JSON array (recommended):

```
"https://aws.amazon.com/roles": [
  "arn:aws:iam::111122223333:role/RoleA",
  "arn:aws:iam::111122223333:role/RoleB"
```

```
]
```

Semicolon-delimited string:

```
"https://aws.amazon.com/roles": "arn:aws:iam::111122223333:role/
RoleA;arn:aws:iam::111122223333:role/RoleB"
```

Availability – This key is present in requests that use the `AssumeRoleWithWebIdentity` operation. The value is `true` when the token includes the `https://aws.amazon.com/roles` claim and the requested role ARN matches a value in the claim. The value is `false` when the token does not include the claim. Because the key is always present in `AssumeRoleWithWebIdentity` requests, use the `Bool` condition operator rather than `BoolIfExists`.

When the token includes the claim but the requested role ARN does not match a value in the claim, the `AssumeRoleWithWebIdentity` call returns an `InvalidIdentityToken` error.

You can use this condition key in role trust policies, resource control policies (RCPs), and VPC endpoint policies to require that an IdP authorize the role before it can be assumed.

The following role trust policy allows `AssumeRoleWithWebIdentity` only when the IdP authorized the requested role in the token.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
idp.example.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": { "idp.example.com:aud": "my-app-id" },
                "Bool": { "sts:RoleAuthorizedByIdp": "true" }
            }
        }
    ]
}
```

sts:RoleSessionName

Works with string operators.

Use this key to compare the session name that a principal specifies when assuming a role with the value that is specified in the policy.

Availability – This key is present in the request when the principal assumes the role using the AWS Management Console, any assume-role CLI command, or any AWS STS `AssumeRole` API operation.

You can use this key in a role trust policy to require that your users provide a specific session name when they assume a role. For example, you can require that IAM users specify their own user name as their session name. After the IAM user assumes the role, activity appears in AWS CloudTrail logs with the session name that matches their user name. This makes it easier for administrators to differentiate between role sessions when a role is used by different principals.

The following role trust policy requires that IAM users in account 111122223333 provide their IAM user name as the session name when they assume the role. This requirement is enforced using the `aws:username` condition variable in the condition key. This policy allows IAM users to assume the role to which the policy is attached. This policy does not allow anyone using temporary credentials to assume the role because the `username` variable is present for only IAM users.

> **Important**
>
> You can use any single-valued condition key as a variable. You can't use a multivalued condition key as a variable.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "RoleTrustPolicyRequireUsernameForSessionName",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
```

```
            "Condition": {
                "StringLike": {"sts:RoleSessionName": "prefix-
${aws:username}"}
            }
        }
    ]
}
```

When an administrator views the AWS CloudTrail log for an action, they can compare the session name to the user names in their account. In the following example, the user named `matjac` performed the operation using the role named `MateoRole`. The administrator can then contact Mateo Jackson, who has the user named `matjac`.

```
"assumedRoleUser": {
    "assumedRoleId": "AROACQRSTUVWRAOEXAMPLE:matjac",
    "arn": "arn:aws:sts::111122223333:assumed-role/MateoRole/matjac"
}
```

If you allow cross-account access using roles, then users in one account can assume a role in another account. The ARN of the assumed role user listed in CloudTrail includes the account *where the role exists*. It does not include the account of the user that assumed the role. Users are unique only within an account. Therefore, we recommend that you use this method for checking CloudTrail logs only for roles that are assumed by users in accounts that you administer. Your users might use the same user name in multiple accounts.

sts:SourceIdentity

Works with string operators.

Use this key to compare the source identity that a principal specifies when assuming a role with the value that is specified in the policy.

Availability – This key is present in the request when the principal provides a source identity while assuming a role using any AWS STS assume-role CLI command, or AWS STS `AssumeRole` API operation.

You can use this key in a role trust policy to require that your users set a specific source identity when they assume a role. For example, you can require your workforce or federated identities to specify a value for source identity. You can configure your identity provider (IdP) to use one of the attributes that are associated with your users, like a user name or email as the source

identity. The IdP then passes the source identity as an attribute in the assertions or claims that it sends to AWS. The value of the source identity attribute identifies the user or application who is assuming the role.

After the user assumes the role, activity appears in AWS CloudTrail logs with the source identity value that was set. This makes it easier for administrators to determine who or what performed actions with a role in AWS. You must grant permissions for the `sts:SetSourceIdentity` action to allow an identity to set a source identity.

Unlike `sts:RoleSessionName`, after the source identity is set, the value cannot be changed. It is present in the request context for all actions taken with the role by the source identity. The value persists into subsequent role sessions when you use the session credentials to assume another role. Assuming one role from another is called role chaining.

You can use the `aws:SourceIdentity` global condition key to further control access to AWS resources based on the value of source identity in subsequent requests.

The following role trust policy allows the IAM user `AdminUser` to assume a role in account `111122223333`. It also grants permission to the `AdminUser` to set a source identity, as long as the source identity set is `DiegoRamirez`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAdminUserAssumeRole",
            "Effect": "Allow",
            "Principal": {"AWS": " arn:aws:iam::111122223333:user/AdminUser"},
            "Action": [
                "sts:AssumeRole",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringEquals": {"sts:SourceIdentity": "DiegoRamirez"}
            }
        }
    ]
}
```

To learn more about using source identity information, see Monitor and control actions taken with assumed roles.

sts:TaskPolicyArn

Works with ARN operators.

Use this key to compare the policy ARN in an sts:AssumeRoot request with the policy ARN specified in the policy.

Availability – This key is present in the request when you make a request using sts:AssumeRoot.

Administrators can use this condition key in IAM policies to restrict specific roles or users within the management account or delegated administrator account from performing certain actions when assuming root credentials. For more information, see Perform a privileged task on an AWS Organizations member account.

sts:TransitiveTagKeys

Works with string operators.

Use this key to compare the transitive session tag keys in the request with those specified in the policy.

Availability – This key is present in the request when you make a request using temporary security credentials. These include credentials created using any assume-role operation, or the `GetFederationToken` operation.

When you make a request using temporary security credentials, the request context includes the `aws:PrincipalTag` context key. This key includes a list of session tags, transitive session tags, and role tags. Transitive session tags are tags that persist into all subsequent sessions when you use the session credentials to assume another role. Assuming one role from another is called role chaining.

You can use this condition key in a policy to require setting specific session tags as transitive when assuming a role or federating a user.

Actions, resources, and condition keys for AWS services

Each AWS service can define actions, resources, and condition context keys for use in IAM policies. For a list of AWS services and their actions, resources, and condition context keys, see Actions, resources, and condition keys in the *Service Authorization Reference.*

Resources to learn more about IAM

IAM is a rich product, and you'll find many resources to help you learn more about how IAM can help you secure your AWS account and resources.

Topics

- Identities
- Credentials (passwords, access keys, and MFA devices)
- Permissions and policies
- Federation and delegation
- IAM and other AWS products
- General security practices
- General resources

Identities

Consult these resources for creating, managing, and using identities.

- Manage identities in IAM Identity Center – Procedural information about creating users and group in IAM Identity Center.
- **IAM Identities** – An in-depth discussion of users, groups, and roles.

Credentials (passwords, access keys, and MFA devices)

Review the following guides to manage passwords, access keys, and MFA devices for your AWS account and for IAM users.

- **User passwords in AWS** – Describes options for managing passwords for IAM users in your account.
- **Manage access keys for IAM users** – Describes how *access keys* work and how you can use them to make programmatic calls to AWS. There are other more secure alternatives to access keys that we recommend you consider first. For more information, see Considerations and alternatives for long-term access keys in the *AWS General Reference guide*.

- **AWS Multi-factor authentication in IAM** – Describes how to configure your account and IAM users to require both a password and a one-time use code that is generated on a device before sign-in is allowed. (This is sometimes called two-factor authentication.)

For general information about the types of credentials you use to access Amazon Web Services, see **AWS Security Credentials** in the *AWS General Reference guide*..

Permissions and policies

Learn the inner workings of IAM policies and find tips on the best ways to confer permissions:

- **Policies and permissions in AWS Identity and Access Management** – Introduces the policy language that is used to define permissions. Describes how permissions can be attached to users or groups or, for some AWS products, to resources themselves.
- **IAM JSON policy element reference** – Provides descriptions and examples of each policy language element.
- **IAM policy validation** – Find resources for JSON policy validation.
- **Example IAM identity-based policies** – Shows examples of policies for common tasks in various AWS products.
- **AWS Policy Generator** – Create custom policies by choosing products and actions from a list.
- **IAM Policy Simulator** – Test whether a policy would allow or deny a specific request to AWS.

Federation and delegation

You can grant access to resources in your AWS account for users who are authenticated (signed in) elsewhere. These can be IAM users in another AWS account (known as *delegation*), users who are authenticated with your organization's sign-in process, or users from an Internet identity provider like Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC) compatible identity provider. In these cases, the users get temporary security credentials to access AWS resources.

- **IAM tutorial: Delegate access across AWS accounts using IAM roles** – Guides you through granting cross-account access to an IAM user in another AWS account.
- **Common scenarios for temporary credentials** – Describes ways in which users can be federated into AWS after being authenticated outside of AWS.

IAM and other AWS products

Most AWS products are integrated with IAM so that you can use IAM features to help protect access to the resources in those products. The following resources discuss IAM and security for some of the most popular AWS products. For a complete list of products that work with IAM, including links to more information on each, see AWS services that work with IAM.

Using IAM with Amazon EC2

- Controlling Access to Amazon EC2 Resources – Describes how to use IAM features to permit users to administer Amazon EC2 instances, volumes, and more.
- Use instance profiles – Describes how to use IAM roles to securely provide credentials for applications that run on Amazon EC2 instances and that need access to other AWS products.

Using IAM with Amazon S3

- Managing Access Permissions to Your Amazon S3 Resources – Discusses the Amazon S3 security model for buckets and objects, which includes IAM policies.
- Writing IAM Policies: Grant Access to User-Specific Folders in an Amazon S3 Bucket – Discusses how to let users protect their own folders in Amazon S3. (For more posts about Amazon S3 and IAM, choose the **S3** tag below the title of the blog post.)

Using IAM with Amazon RDS

- Using AWS Identity and Access Management (IAM) to Manage Access to Amazon RDS Resources – Describes how to use IAM to control access to database instances, database snapshots, and more.
- A Primer on RDS Resource-Level Permissions – Describes how to use IAM to control access to specific Amazon RDS instances.

Using IAM with Amazon DynamoDB

- Using IAM to Control Access to DynamoDB Resources – Describes how to use IAM to permit users to administer DynamoDB tables and indexes.
- The following video (8:55) explains how to provide access control for individual DynamoDB database items or attributes (or both).

Getting Started with Fine-Grained Access Control for DynamoDB

General security practices

Find expert tips and guidance on the best ways to secure your AWS account and resources:

- **Best Practices for Security, Identity, &, Compliance** – Find resources for how to manage security across AWS accounts and products, including suggestions for security architecture, use of IAM, encryption and data security, and more.
- **Identity and Access Management** – The AWS Well-Architected Framework helps you understand key concepts, design principles, and architectural best practices for designing and running workloads in the cloud.
- **Security best practices in IAM** – Offers recommendations for ways to use IAM to help secure your AWS account and resources.
- **AWS CloudTrail User Guide** – Use AWS CloudTrail to track a history of API calls made to AWS and store that information in log files. This helps you determine which users and accounts accessed resources in your account, when the calls were made, what actions were requested, and more.

General resources

Explore the following resources to learn more about IAM and AWS.

- **Product Information for IAM** – General information about the AWS Identity and Access Management product.
- **AWS re:Post for AWS Identity and Access Management** – Visit AWS re:Post to discuss technical questions related to IAM with the AWS community.

- Classes & Workshops – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.
- AWS Developer Center – Explore tutorials, download tools, and learn about AWS developer events.
- AWS Developer Tools – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.

- Getting Started Resource Center – Learn how to set up your AWS account, join the AWS community, and launch your first application.
- Hands-On Tutorials – Follow step-by-step tutorials to launch your first application on AWS.
- AWS Whitepapers – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- AWS Support Center – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- Support – The primary webpage for information about Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- Contact Us – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- AWS Site Terms – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Calling the IAM API using HTTP query requests

Contents

- Endpoints
- HTTPS required
- Signing IAM API requests

You can access the IAM and AWS STS services programmatically using the Query API. Query API requests are HTTPS requests that must contain an `Action` parameter to indicate the action to be performed. IAM and AWS STS support GET and POST requests for all actions. That is, the API does not require you to use GET for some actions and POST for others. However, GET requests are subject to the limitation size of a URL; although this limit is browser dependent, a typical limit is 2048 bytes. Therefore, for Query API requests that require larger sizes, you must use a POST request.

The response is an XML document. For details about the response, see the individual action pages in the IAM API Reference or the AWS Security Token Service API Reference.

> **Tip**
>
> Instead of making direct calls to the IAM or AWS STS API operations, you can use one of the AWS SDKs. The AWS SDKs consist of libraries and sample code for various programming languages and platforms (Java, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests (see below), managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see the Tools for Amazon Web Services page.

For details about the API actions and errors, see the IAM API Reference or the AWS Security Token Service API Reference.

Endpoints

IAM and AWS STS each have a single global endpoint:

- (IAM) https://iam.amazonaws.com
- (AWS STS) https://sts.amazonaws.com

> **Important**
>
> AWS STS also supports sending requests to regional endpoints in addition to the global endpoint. AWS recommends using regional endpoints instead of global endpoints to reduce latency, build in redundancy, and increase session token validity. Before you can use AWS STS in a Region, you must first activate STS in that Region for your AWS account. For more information about activating additional Regions for AWS STS, see Manage AWS STS in an AWS Region.

For more information about AWS endpoints and Regions for all services, see Service endpoints and quotas in the *AWS General Reference*.

HTTPS required

Because the Query API returns sensitive information such as security credentials, you must use HTTPS with all API requests.

Signing IAM API requests

Requests must be signed using an access key ID and a secret access key. We strongly recommend that you do not use your AWS account root user credentials for everyday work with IAM. You can use the credentials for an IAM user or you can use AWS STS to generate temporary security credentials.

To sign your API requests, we recommend using AWS Signature Version 4. For information about using Signature Version 4, see Signature Version 4 Signing Process in the *AWS General Reference*.

If you need to use Signature Version 2, information about using Signature Version 2 is available in the AWS General Reference.

For more information, see the following:

- AWS Security Credentials. Provides general information about the types of credentials used for accessing AWS.

- Security best practices in IAM. Presents a list of suggestions for using IAM service to help secure your AWS resources.
- Temporary security credentials in IAM. Describes how to create and use temporary security credentials.

Document history for IAM

The following table describes major documentation updates for IAM.

Change	Description	Date
job_workflow_ref accepted as alternative scoping claim for GitHub Actions OIDC provider	For the GitHub Actions OIDC provider, IAM now accepts either the sub claim or the `job_workflow_ref` claim as the identity-provider control in role trust policies.	May 12, 2026
IAM condition context keys for service-specific credential APIs	IAM now supports two new condition context keys for controlling access to service-specific credential APIs: `iam:ServiceSpecificCredentialAgeDays` and `iam:ServiceSpecificCredentialServiceName`. These keys allow you to restrict the creation and management of service-specific credentials based on expiration settings and permitted AWS services. • `iam:ServiceSpecificCredentialAgeDays` • `iam:ServiceSpecificCredentialServiceName`	September 4, 2025
VPC endpoint condition keys for network perimeter controls	IAM now supports three new AWS global condition context keys for implementing	August 28, 2025

	scalable network perimeter controls: `aws:VpceAccount` , `aws:VpceOrgID` , and `aws:VpceOrgPaths` . These keys help ensure requests come through VPC endpoints owned by specific accounts, organizations, or organizational units, automatically scaling with your VPC endpoint usage without requiring policy updates when you create new endpoints. For more information, see Establish permissions guardrails using data perimeters.	
IAM SAML federation CloudFormation tutorials	IAM added new tutorials for creating SAML identity providers (IdPs) and federated roles using CloudFormation. • Create SAML IdP with CloudFormation • Create SAML federated role with CloudFormation • Create SAML IdP and federated role with CloudFormation	August 25, 2025

Additional shared OIDC provider controls for shared OIDC providers	IAM now includes Amazon Cognito, Azure Sentinel, Pulumi Cloud, and Vercel global endpoint in the list of shared OIDC identity providers that require explicit evaluation of specific claims in JSON Web Tokens (JWTs).	August 1, 2025
IAM Access Analyzer added internal access analyzers	IAM Access Analyzer helps you identify which principal s within your organization or account have access to selected business-critical resources. Internal access analyzers support implement ing the principle of least privilege by ensuring that your specified resources can only be accessed by the intended principals within your AWS organization or account.	June 16, 2025
Identity provider controls for shared OIDC providers	IAM now requires explicit evaluation of specific claims in JSON Web Tokens (JWTs) for recognized shared OIDC identity providers. This security control ensures that only authorized identities from the intended organizat ion can assume roles and access AWS resources.	June 6, 2025

IAMUserChangePassword – Added permissions	IAM added permissions to IAMUserChangePassword to allow users specified within a path.	May 28, 2025
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added `iam:GetAccountAuthorizationDetails` to the service-level permissions of AccessAnalyzerServiceRolePolicy.	May 12, 2025
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added Amazon S3 directory bucket access points to the service-level permissions of AccessAnalyzerServiceRolePolicy.	March 31, 2025
IAM dual-stack endpoint support	IAM now provides improved dual-stack endpoint support that enables clients to communicate with IAM using either IPv4 or IPv6 addresses.	March 20, 2025
Updates to policy evaluation logic	Enhanced documentation for policy evaluation logic, including improved flow charts and clearer explanations of how AWS evaluates policies to determine whether to allow or deny a request.	January 30, 2025
IAMDeleteRootUserCredentials – Removed permissions	IAM removed the `iam:DeleteVirtualMFADevice` permission from the managed policy.	January 7, 2025

AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve information about Amazon ECR account settings and registry policies to the service-level permissions of AccessAnalyzerServiceRolePolicy.	December 10, 2024
IAM Access Analyzer added access configuration	IAM Access Analyzer added support to configure analyzers to change the scope of which AWS accounts, IAM users, and roles generate findings.	November 14, 2024
Centrally manage root access for member accounts	You can now manage privileged root user credentials across member accounts in AWS Organizations with centralized root access. Centrally secure the root user credentials of your AWS accounts managed using AWS Organizations to remove and prevent root user credential recovery and access at scale. • Root user best practices	November 14, 2024

AWS managed policy update - New policies	IAM added two new policies to scope permissions for privileged root user sessions that you can initiate after you centralize root user access for member accounts in your organization. • IAMAuditRootUserCredentials • IAMCreateRootUserPassword • IAMDeleteRootUserCredentials • S3UnlockBucketPolicy • SQSUnlockQueuePolicy	November 14, 2024
Support for AWS Organizations resource control policies (RCPs)	Use an AWS Organizations resource control policy (RCP) to define the maximum permissions for resources within accounts in your organization or organizational unit (OU). RCPs limit permissions that identity-based and resource-based policies can grant to resources in accounts within your organization.	November 13, 2024

AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve information about IAM user and role tags to the service-level permissions of AccessAnalyzerServiceRolePolicy.	October 29, 2024
SAML encryption support enhancements	Enhanced documentation for SAML encryption support in IAM SAML providers, including improved troubleshooting guidance and clarification on service compatibility.	June 5, 2024
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve information about IAM user and role policies to the service-level permissions of AccessAnalyzerServiceRolePolicy.	May 30, 2024
Encryption support for SAML identity providers	IAM SAML providers now support encrypted assertions in the SAML response from your external IdP. To understand how encryption works with IAM SAML federation, see Using SAML-based federation for API access.	February 4, 2024

AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added support for permission to retrieve the current state of the block public access for Amazon EC2 snapshots to the service-level permissions of AccessAnalyzerServiceRolePolicy.	January 23, 2024
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added DynamoDB streams and tables to the service-level permissions of AccessAnalyzerServiceRolePolicy.	January 11, 2024
AccessAnalyzerServiceRolePolicy – Added permissions	IAM Access Analyzer added Amazon S3 directory buckets to the service-level permissions of AccessAnalyzerServiceRolePolicy.	December 1, 2023
IAMAccessAnalyzerReadOnlyAccess – Added permissions	IAM Access Analyzer added permissions to IAMAccessAnalyzerReadOnlyAccess to allow you to check whether updates to your policies grant additional access. This permission is required by IAM Access Analyzer to perform policy checks on your policies.	November 26, 2023

IAM Access Analyzer added unused access analyzers	IAM Access Analyzer simplifies inspecting unused access to guide you toward least privilege. IAM Access Analyzer continuously analyzes your accounts to identify unused access and creates a centralized dashboard with findings.	November 26, 2023
IAM Access Analyzer added custom policy checks	IAM Access Analyzer now provides custom policy checks to validate that IAM policies adhere to your security standards ahead of deployments.	November 26, 2023
`AccessAnalyzerServiceRolePolicy` – Added permissions	IAM Access Analyzer added IAM actions to the service-level permissions of AccessAnalyzerServiceRolePolicy to support the following actions: • Listing entities for a policy • Generating service last accessed details • Listing access key information	November 26, 2023
Action last accessed information and policy generation support for over 60 additional services and actions	IAM now supports action last accessed information and generates policies with action-level information for over 60 additional services, along with a list of the actions for which action last accessed information is available.	November 1, 2023

Action last accessed information support for over 140 services	IAM now provides action last accessed information for more than 140 services, along with a list of the actions for which action last accessed information is available.	September 14, 2023
Support for multiple multi-factor authentication (MFA) devices for root users and IAM users	Now you can to add up to eight MFA devices per user, including FIDO security keys, software time-based one-time password (TOTP) with virtual authenticator applications, or hardware TOTP tokens.	November 16, 2022
IAM Access Analyzer support for new resource types	IAM Access Analyzer added support for the following resource types: • Amazon EBS volume snapshots • Amazon ECR repositories • Amazon EFS file systems • Amazon RDS DB snapshots • Amazon RDS DB cluster snapshots • Amazon SNS topics	October 25, 2022
U2F deprecation and WebAuthn/FIDO update	Removed mentions of U2F as an MFA option and added information about WebAuthn, FIDO2, and FIDO security keys.	May 31, 2022

Updates to resilience in IAM	Added information about maintaining access to IAM credentials when an event disrupts communication between AWS Regions.	May 16, 2022
New global condition keys for resources	You can now control access to resources based on the account, Organizational Unit (OU), or organization in AWS Organizations that contains your resources. You can use the `aws:ResourceAccount` , `aws:ResourceOrgID` , and `aws:ResourceOrgPaths` global condition keys in an IAM policy.	April 27, 2022
Code examples for IAM using AWS SDKs	Added code examples that show how to use IAM with an AWS software development kit (SDK). The examples are divided into code excerpts that show you how to call individual service functions and examples that show you how to accomplish a specific task by calling multiple functions within the same service.	April 7, 2022
Updates to policy evaluation logic flow chart	Updates to the policy evaluation logic flow chart and related text in the Determining whether a request is allowed or denied within an account section.	November 17, 2021

Updates to security best practices	Added information about creating administrative users instead of using root user credentials, removed the best practice of using IAM groups to assign permissions to IAM users, and clarified when to use managed policies instead of inline policies.	October 5, 2021
Updates to policy evaluation logic topic for resource-based policies	Added information about the impact of resource-based policies and different principal types in the same account.	October 5, 2021
Updates to single-valued and multivalued condition keys	The differences between single-valued and multivalued condition keys are now explained in more detail. The value type was added to each AWS global condition context key.	September 30, 2021
IAM Access Analyzer supports Amazon S3 Multi-Region Access Points	IAM Access Analyzer identifies Amazon S3 buckets that allow public and cross-account access, including those that use Amazon S3 Multi-Region Access Points.	September 2, 2021
AWS managed policy updates - Update to an existing policy	IAM Access Analyzer updated an existing AWS managed policy.	September 2, 2021

More services supported for action-level policy generation	IAM Access Analyzer can generate IAM policies with action-level access activity information for additional AWS services.	August 24, 2021
Generate IAM policies for cross-account trails	You can now use IAM Access Analyzer to generate fine-grai ned policies based on your access activity using a AWS CloudTrail trail in a different account, for example, a centralized AWS Organizat ions trail.	August 18, 2021

Additional IAM Access Analyzer policy checks	IAM Access Analyzer extended policy validation by adding new policy checks that validate conditions included in IAM policies. These checks analyze the condition block in your policy statement and report security warnings, errors, and suggestions along with actionable recommend ations. IAM Access Analyzer added the following policy checks: • Error – Invalid service principal format • Error – Missing tag key in condition • Security Warning – Deny NotAction with unsupport ed tag condition key for service • Security Warning – Deny with unsupported tag condition key for service • Security Warning – Missing paired condition keys • Suggestion – Allow NotAction with unsupport ed tag condition key for service • Suggestion – Allow with unsupported tag condition key for service	June 29, 2021

Action last accessed support for more services	You can now view action last accessed information in the IAM console about the last time an IAM principal used an action for the following services: Amazon EC2, IAM, Lambda, and Amazon S3 management actions. You can also use the AWS CLI or AWS API to retrieve a data report. You can use this informati on to identify unnecessary permissions so that you can refine your IAM policies to better adhere to the principle of least privilege.	April 19, 2021
Monitor and control actions taken with assumed roles	Administrators can configure IAM roles to require that identities pass a source identity, which is logged in AWS CloudTrail. Reviewing source identity informati on helps administrators determine who or what performed actions with assumed role sessions.	April 13, 2021
Generate IAM policies based on access activity	You can now use IAM Access Analyzer to generate fine-grai ned policies based on your access activity found in your AWS CloudTrail.	April 7, 2021

IAM Access Analyzer policy checks	IAM Access Analyzer now provides over 100 policy checks with actionable recommendations during policy authoring.	March 16, 2021
Expanded policy validation options	Expanded policy validation available in the IAM console, AWS API, and AWS CLI using policy checks in IAM Access Analyzer to help you author secure and functional JSON policies.	March 15, 2021
Tagging IAM resources	You can now tag additional IAM resources using a tag key-value pair.	February 11, 2021
Default password policy for IAM users	If you do not set a custom password policy for your AWS account, IAM user passwords must now meet the default AWS password policy.	November 18, 2020
The actions, resources, and condition keys pages for AWS services have moved	Each AWS service can define actions, resources, and condition context keys for use in IAM policies. You can now find the list of AWS services and their actions, resources , and condition context keys in the *Service Authorization Reference.*	November 16, 2020

IAM users longer role session duration	IAM users can now have a longer role session duration when switching roles in the AWS Management Console, reducing interruptions due to session expiration. Users are granted the maximum session duration set for the role, or the remaining time in the IAM user's session, whichever is less.	July 24, 2020
Use Service Quotas to request quick increases for IAM entities	You can request quota increases for adjustable IAM quotas using the Service Quotas console. Now, some increases are automatically approved in Service Quotas and available in your account within a few minutes. Larger requests are submitted to AWS Support.	June 25, 2020

Last accessed information in IAM now includes Amazon S3 management actions	In addition to service last accessed information, you can now view information in the IAM console about the last time an IAM principal used an Amazon S3 action. You can also use the AWS CLI or AWS API to retrieve the data report. The report includes information about the allowed services and actions that principals last attempted to access and when. You can use this information to identify unnecessary permissions so that you can refine your IAM policies to better adhere to the principle of least privilege.	June 3, 2020
Security chapter addition	The security chapter helps you understand how to configure IAM and AWS STS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your IAM resources.	April 29, 2020
sts:RoleSessionName	You can now write a policy that grants permissions based on the session name that a principal specifies when assuming a role.	April 21, 2020

AWS sign-in page update	When you sign in on the main AWS sign-in page, you can now choose to sign in as the AWS account root user or an IAM user. When you do, the label on the page indicates whether you should provide your root user email address or your IAM user information. This documentation includes updated screen captures to help you understand the AWS sign-in pages.	March 4, 2020
aws:ViaAWSService and aws:CalledVia condition keys	You can now write a policy to limit whether services can make requests on behalf of an IAM principal (user or role). When a principal makes a request to an AWS service, that service might use the principal's credentials to make subsequent requests to other services. Use the `aws:ViaAWSService` condition key to match if any service makes a request using a principal's credentia ls. Use the `aws:CalledVia` condition keys to match if specific services make a request using a principal's credentials.	February 20, 2020

Policy simulator adds support for permissions boundaries	You can now test the effect of permissions boundaries on IAM entities with the IAM policy simulator.	January 23, 2020
Cross-account policy evaluation	You can now learn how AWS evaluates policies for cross-account access. This occurs when a resource in a trusting account includes a resource-based policy that allows a principal in another account to access the resource. The request must be allowed in both accounts.	January 2, 2020
Session tags	You can now include tags when you assume a role or federate a user in AWS STS. When you perform the `AssumeRole` or `GetFederationToken` operation, you can pass the session tags as attributes. When you perform the `AssumeRoleWithSAML` or `AssumeRoleWithWebIdentity` operations, you can pass attributes from your corporate identities to AWS.	November 22, 2019

Control access for groups of AWS accounts in AWS Organizations	You can now reference organizational units (OUs) from AWS Organizations in IAM policies. If you use AWS Organizations to organize your accounts into OUs, you can require that principal s belong to a specific OU before granting access to your resources. Principals include AWS account root user, IAM users and IAM roles. To do this, specify the OU path in the `aws:PrincipalOrgPaths` condition key in your policies.	November 20, 2019
Role last used	You can now view the date, time, and Region where a role was last used. This information also helps you identify unused roles in your account. You can use the AWS Management Console, AWS CLI and AWS API to view information about when a role was last used.	November 19, 2019

Update to the global condition context keys page	You can now learn when each of the global condition keys is included in the context of a request. You can also navigate to each key more easily using the page table of contents (TOC). The information on the page helps you to write more accurate policies. For example, if your employees use federation with IAM roles, you should use the `aws:userId` key and not the `aws:userName` key. The `aws:userName` key applies only to IAM users and not roles.	October 6, 2019
ABAC in AWS	Learn how attribute-based access control (ABAC) works in AWS using tags, and how it compares to the tradition al AWS authorization model. Use the ABAC tutorial to learn how to create and test a policy that allows IAM roles with principal tags to access resources with matching tags. This strategy allows individua ls to view or edit only the AWS resources required for their jobs.	October 3, 2019

AWS STS GetAccessKeyInfo operation	You can review the AWS access keys in your code to determine whether the keys are from an account that you own. You can pass an access key ID using the `aws sts get-access-key-info` AWS CLI command or the `GetAccessKeyInfo` AWS API operation.	July 24, 2019
Viewing AWS Organizations service last accessed information in IAM	You can now view service last accessed information for an AWS Organizations entity or policy in the **AWS Organizations** section of the IAM console. You can also use the AWS CLI or AWS API to retrieve the data report. This data includes information about the allowed services that principals in an AWS Organizations account last attempted to access and when. You can use this information to identify unnecessary permissions so that you can refine your AWS Organizations policies to better adhere to the principle of least privilege.	June 20, 2019

Using a managed policy as a session policy	You can now pass up to 10 managed policy ARNs when you assume a role. This allows you to limit the permissio ns of the role's temporary credentials.	May 7, 2019
AWS STS Region compatibi lity of session tokens for the global endpoint	You can now choose whether to use version 1 or version 2 global endpoint tokens. Version 1 tokens are valid only in AWS Regions that are available by default. These tokens will not work in manually enabled Regions, such as Asia Pacific (Hong Kong). Version 2 tokens are valid in all Regions. However, version 2 tokens are longer and might affect systems where you temporarily store tokens.	April 26, 2019
Allow enabling and disabling AWS regions	You can now create a policy that allows an administrator to enable and disable the Asia Pacific (Hong Kong) Region (ap-east-1).	April 24, 2019

IAM user my security credentials page	IAM users can now manage all of their own credentials on the **My Security Credentials** page. This AWS Managemen t Console page displays account information such as the account ID and canonical user ID. Users can also view and edit their own passwords, access keys, X.509 certificates, SSH keys, and Git credentials.	January 24, 2019
Access Analyzer API	You can now use the AWS CLI and AWS API to view service last accessed information.	December 7, 2018
Tagging IAM users and roles	You can now use IAM tags to add custom attributes to an identity (IAM user or role) using a tag key-value pair. You can also use tags to control an identity's access to resources or to control what tags can be attached to an identity.	November 14, 2018
U2F security keys	You can now use U2F security keys as a multi-factor authentication (MFA) option when signing in to the AWS Management Console.	September 25, 2018
Support for Amazon VPC endpoints	You can now establish a private connection between your VPC and AWS STS in the US West (Oregon) Region.	July 31, 2018

Permissions boundaries	New feature makes it easier to grant trusted employees the ability to manage IAM permissions without also granting full IAM administr ative access.	July 12, 2018
aws:PrincipalOrgID	New condition key provides an easier way to control access to AWS resources by specifying the AWS organizat ion of IAM principals.	May 17, 2018
aws:RequestedRegion	New condition key provides an easier way to use IAM policies to control access to AWS Regions.	April 25, 2018
Increased session duration for IAM roles	An IAM role can now have a session duration of 12 hours.	March 28, 2018
Updated role-creation workflow	New workflow improves the process of creating trust relationships and attaching permissions to roles.	September 8, 2017
AWS account sign-in process	Updated AWS sign-in experience allows both the root user and IAM users to use the **Sign In to the Console** link on the AWS Management Console's home page.	August 25, 2017
Example IAM policies	Documentation update features more than 30 example policies.	August 2, 2017

IAM best practices	Information added to the **Users** section of the IAM console makes it easier to follow IAM best practices.	July 5, 2017
Auto Scaling resources	Resource-level permissions can control access to and permissions for Auto Scaling resources.	May 16, 2017
Amazon RDS for MySQL and Amazon Aurora databases	Database administrators can associate database users with IAM users and roles and thus manage user access to all AWS resources from a single location.	April 24, 2017
Service-linked roles	Service-linked roles provide an easier and more secure way to delegate permissions to AWS services.	April 19, 2017
Policy summaries	New policy summaries make it easier to understand permissions in IAM policies.	March 23, 2017